**Львівський національний аграрний університет Факультет механіки та енергетики**

## кафедра інформаційних систем та технологій

**КУРСОВА РОБОТА**

**з дисципліни «Об’єктно-орієнтоване програмування»**

на тему: «Розроблення системи підтримки прийняття рішень в управлінні персоналом»

Студент 1 курсу КН-12сп групи денної форми навчання механіки та енергетики факультету

Павлюх Павло Ігорович

Науковий керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

к. т. н., в. о. доцента, Татомир А.В.

Дубляни 2022

**ЗМІСТ**

ВСТУП 3

РОЗДІЛ 1. АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАВДАННЯ НА РОБОТУ 4

1.1. Аналіз предмету проектування 4  
1.2. Класифікація об’єктно-орієнтованих мов програмування 4  
1.3. Огляд та аналіз сучасних технологій та засобів проектування програмного забезпечення 8  
1.4. Універсальна мова проектування UML 9

1.5. Уточнена постановка задачі на розробку програмного забезпечення 10

РОЗДІЛ 2. РОЗРОБКА ЗАГАЛЬНОЇ СТРУКТУРИ ПРОГРАМИ 13

2.1. Аналіз функцій системи 13

2.2. Розроблення структурної моделі системи 13

РОЗДІЛ 3. РОЗРОБЛЕННЯ ІЄРАРХІЇ КЛАСІВ 16

3.1. Декомпозиція системи 16

3.2. Аналіз взаємозв’язків між об’єктами 16

3.3. Розроблення інтерфейсів класів 17

3.4. UML-діаграма класів 17

РОЗДІЛ 4. РОЗРОБКА СИСТЕМИ ВВОДУ-ВИВОДУ ДАНИХ 20

РОЗДІЛ 5. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 22

РОЗДІЛ 6. РОЗРОБКА ДОКУМЕНТАЦІЇ ДЛЯ СУПРОВОДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ 27

ВИСНОВКИ 29

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 30

**ВСТУП**

Об’єктно-орієнтоване програмування (ООП) – це еволюційний крок, який випливає із розвитку програмування. ООП дає нам можливість відчути себе не тільки програмістом, а й архітектором, проектуючи структуру програми, створюючи красиві форми.

Ціллю є демонстрація основи ООП на прикладі програми «Система підтримки прийняття рішень в управлінні персоналом».

В сучасному світі людство оперує безмежною кількістю інформації, яку зберігати та сортувати без допоміжних засобів просто неможливо. Тому актуальність систем управління сьогодні дуже висока. Вони допомагають упорядковувати, знаходити та порівнювати певну структуру даних.

Задача розробки системи підтримки прийняття рішень також вимагає створення системи управління, щоб можна було легко, а головне, швидко знайти потрібного працівника, створити нового, редагувати інформацію про нього тощо.

**РОЗДІЛ 1. АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАВДАННЯ НА РОБОТУ**

* 1. **Аналіз предмету проектування**

Предметом проектування є Система підтримки прийняття рішень в управлінні персоналом.

Система підтримки рішень — комп'ютеризована система, яка через збирання та аналіз великого обсягу інформації може впливати на процес ухвалення керівничих рішень в бізнесі та підприємництві.

Інформаційна система — сукупність організаційних і технічних засобів для збереження та [обробки інформації](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%97) з метою забезпечення інформаційних потреб користувачів.

* 1. **Класифікація об’єктно-орієнтованих мов програмування**

Класифікацій мов програмування існує багато, але наукової теорії поки що немає. Три основні класифікації: за функціональною силою, за предметною орієнтацією, за рівнем абстракції.

За функціональною силою:

1. Універсальні мови ( в них можна промоделювати, умовно кажучи, будь-який алгоритм).
2. Спеціалізовані мови (орієнтовані на певні класи задач).

За предметною орієнтацією:

Кожна мова програмування виникла в процесі розв’язання певного класу задач, наприклад, мови програмування для розв’язання задач символьної обробки (Lisp, Cobol, тощо)

За рівнем абстракції*:*

1. Мови низького рівня (машинно-залежні) – Assembler і т.п.
2. Мови високого рівня (орієнтовані на користувача до певної міри) - Pascal, C,  Fortran, тощо.

Мови програмування низького рівня- орієнтовані на конкретний тип процесора і враховують його особливості.

Переваги:

* з допомогою мов низького рівня створюються ефективні і компактні програми, оскільки розробник отримує доступ до всіх можливостей процесора.

Недоліки:

* програміст, що працює з мовами низького рівня, має бути високої кваліфікації, добре розуміти будову комп’ютера;
* результуюча програма не може бути перенесена на комп’ютер з іншим типом процесора.

Асемблер

Мова Асемблера поєднує в собі достоїнства мови машинних команд і деякі риси мов високого рівня. Асемблер забезпечує можливість застосування символічних імен у вихідній програмі й рятує програміста від стомлюючої праці (неминучого при програмуванні мовою машинних команд) по розподілі пам'яті комп'ютера для команд, змінних і констант. Асемблер дозволяє також гнучко й повно використати технічні можливості комп'ютера, як і мова машинних команд. Транслятор вихідних програм в Асемблері простіше транслятора, що вимагається для мови програмування високого рівня. На Асемблері можна написати настільки ж ефективну за розміром й часом виконання програму, як і програму мовою машинних команд. Ця перевага відсутня в мов високого рівня. Цю мову часто застосовують для програмування систем реального часу, технологічними процесами обладнання, забезпечення роботи інформаційно-вимірювальних комплексів. До таких систем звичайно пред'являються високі вимоги за обсягом займаної машинної пам'яті. Часто мова Асемблера доповнюється засобами формування макрокоманд, кожна з яких еквівалентна цілій групі машинних команд. Таку мову називають мовою макроасемблера.

Мови програмування високого рівня.

Мови програмування високого рівня дозволяють писати програми в формі, більш наближеній до звичайної мови. Програму, написаною мовою високого рівня, можна більш легко читати і модифікувати, і вони значно полегшують роботу програміста порівняно з написанням машинного коду. Для перекладу програм, написаних мовою високого рівня, в машинні коди, повинні існувати спеціальні програми. Такі програми називаються трансляторами. Важливою особливістю мов високого рівня є їх відносна незалежність від машини. Це означає, що правила запису програм не залежать або мало залежать від особливостей конкретної машини. Тоді для перенесення програми на іншу машину програму не обов'язково переписувати заново, достатньо лише відтранслювати її в коди, специфічні для цієї машини. В крайньому разі, зміни в програмі повинні бути мінімальними.

С++

Мова C++ з'явився на початку 80-х років.

Створений Б’єрном Страуструпом з первісною метою позбавити себе й своїх друзів від програмування на асемблері, Сі або різних інших мовах високого рівня.

Очевидно, що найбільше C++ запозичив з мови Сі, а також з безпосереднього його попередника мови BCPL.

Ці запозичення забезпечили C++ потужними засобами низького рівня, що дозволяють вирішувати складні завдання системного програмування. Але що в першу чергу відрізняє C++ від Сі - це різний ступінь уваги до типів і структур даних. Це пов'язано з появою понять класу, похідного класу й віртуальної функції, перейнятих у свою чергу з мови Симула 67.

Це дає в C++ більш ефективні можливості для контролю типів і забезпечує модульність програми.

На думку автора мови, розходження між ідеологією С й C++ полягає приблизно в наступному: програма на С відображає “спосіб мислення” процесора, а C++ - спосіб мислення програміста. Відповідаючи вимогам сучасного програмування, C++ наголошує на розробці нових типів даних найбільше повно відповідним концепціям обраної області знань. Клас є ключовим поняттям C++. Опис класу містить опис даних, що вимагаються для подання об'єктів цього типу й набір операцій для роботи з подібними об'єктами.

На відміну від традиційних структур С й Паскаля, членами класу є не тільки дані, але й функції. Функції - члени класу мають привілейований доступ до даних усередині об'єктів цього класу й забезпечують інтерфейс між цими об'єктами й іншою програмою. При подальшій роботі зовсім не обов'язково пам'ятати про внутрішню структуру класу й механізм роботи убудованих функцій. У цьому змісті клас подібний до електричного приладу - мало хто знає про його пристрій, але всі знають, як ним користуватися.

Мова С++ є засобом об'єктного програмування, новітньої методики проектування й реалізації програм, що у поточному десятилітті, швидше за все, замінить традиційне процедурне програмування.

Головною метою творця мови доктора Б’єрна Страустрапа було оснащення мови С++ конструкціями, що дозволяють збільшити продуктивність праці програмістів і полегшити процес оволодіння великими програмними продуктами.

Абстракція, реалізація, спадкування й поліморфізм є необхідними властивостями якими володіє мова С++, завдяки чому він не тільки універсальний, як і мова С, але і є об'єктною мовою.

Java

Мова Java зародилась як частина проекту створення передового програмного забезпечення (ПО) для різних побутових приладів. Реалізація проекту була почата мовою С++, але незабаром виник ряд проблем, найкращим засобом боротьби з якими була зміна самого інструмента - мови програмування. Стало очевидним, що необхідно платформо-незалежна мова програмування, що дозволяє створювати програми, які не доводилося б компілювати окремо для кожної архітектури й можна було б використати на різних процесорах під різними операційними системами.

Три ключових елементи об'єдналися в технології мови Java:

1. Java надає для широкого використання свої аплети (applets) — невеликі, надійні, динамічні, що не залежать від платформи активні мережні додатки, що вбудовують у сторінки Web. Аплети Java можуть настроюватися й поширюватися споживачам з такою же легкістю, як будь-які документи HTML.
2. Java вивільняє міць об’єктно-орієнтованої розробки додатків, сполучаючи простий і знайомий синтаксис із надійним й зручним в роботі з середовищем розробки. Це дозволяє широкому колу програмістів швидко створювати нові програми й нові аплети.
3. Java надає програмістові багатий набір класів об'єктів для ясного абстрагування багатьох системних функцій, використовуваних при роботі з вікнами, мережею й для вводу-виводу. Ключова риса цих класів полягає в тім, що вони забезпечують створення незалежних від використовуваної платформи абстракцій для широкого спектра системних інтерфейсів.

**1.3 Огляд та аналіз сучасних технологій та засобів проектування програмного забезпечення**

Комп'ютерна система управління (КСУ) — автоматизована система, що ґрунтується на комплексному використанні технічних, математичних, інформаційних та організаційних засобів для управління складними технічними й економічними об'єктами. КСУ - це сукупність керованого об’єкта й автоматичних вимірювальних та керуючих пристроїв, у якій частину функцій виконує людина.

Створені за тридцятилітню історію впровадження ЕОМ у сферу управлінської діяльності численні КСУ різняться призначенням, проблемною орієнтацією, місцем застосування, автоматизованими функціями і т. ін. З метою підвищення ефективності витрат на розвиток діючих систем та проектування нових, усунення паралелізму і дублювання в проведенні наукових досліджень і проектно-конструкторських робіт, створення типових проектних рішень і типових КСУ зроблено їх класифікацію.

КСУ дає змогу розв'язувати задачі перспективного та оперативного планування виробництва, оперативного розподілу завантаження обладнання, оптимального розподілу обладнання та використання ресурсів і інше. АСК належить до класу людино-машинних систем і складається з функціональної і забезпечувальної частин.

Функціональна частина КСУ включає систему моделей планово-економічних і управлінських задач, забезпечувальна частина — інформаційну і технічну бази, математичне забезпечення, економіко-організаційну базу та інше.

Спеціальне математичне забезпечення включає пакети прикладних програм, що здійснюють організацію й обробку даних з метою реалізації необхідних функцій управління в рамках певних економіко-математичних та організаційних моделей. Програмне забезпечення КСУ (ПЗ) містить сукупність програм на носіях, даних і програмних документів, яка призначена для відлагодження, функціонування й перевірки роботоздатності КСУ.

Під час організації процесу проектування, автоматизації виконання проектних робіт застосовують такі засоби:

* технічні *—* утворюють локальну обчислювальну мережу (ЛОМ) процесу проектування;
* мовні *—* вживаються для формального опису задач і забезпечують різні способи відображення інформаційних входів, виходів та алгоритмів їх перетворення.

Під час проектування ПЗ використовують процедурні і непроцедурні мови. Мовні засоби високого рівня непроцедурного типу застосовують як формальні засоби для забезпечення однозначності й можливості аналізу ПЗ. Наприклад, на передпроектній стадії для аналізу результатів обстеження ІС використовують мовні засоби, що ґрунтуються на апараті теорії відношень для специфікації структурних властивостей та алгоритмічних зв'язків компонентів ПЗ.

* 1. **Універсальна мова проектування UML**

UML (англ. *Unified Modeling Language*) — уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної *UML-моделлю*. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація. Перша версія (1.0) UML вийшла 13 січня 1997, вона була створена за запитом ObjectManagementGroup (OMG) — організації, відповідальної за прийняття стандартів в галузі об'єктних технологій і баз даних. Після обговорення, у вересні 1997 року, версія 1.1 UML була представлена на голосування в OMG. Розробку UML підтримали і вже тоді використовували як стандарт такі гранди ринку інформаційних технологій, як Microsoft, IBM, Hewlett-Packard, Oracle,DEC, Sybase, Logic Works й інші.

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем. Діаграми дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код.

Основною причиною використання мови UML є спілкування розробників між собою.

Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації у кілька разів і помітно поліпшити якість кінцевого продукту.

UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об’єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки.

Практично усі CASE-засоби (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи.

Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

* 1. **Уточнена постановка задачі на розробку програмного забезпечення**

На основі виконаного аналізу та огляду літературних джерел можна сформулювати постановку задачі.

Організаційна сутність задачі

Розробити програмне забезпечення інформаційної системи «Система підтримки прийняття рішень в управлінні персоналом», основними функціями якої є: можливість додавання працівників користувачем, можливість пошуку та перегляду всіх працівників системи, зміна та видалення даних існуючих працівників, можливість задання ієрархічної структури (одні працівники можуть підпорядковуватися іншим), можливість фільтрування за різними показниками (функціональні обов’язки, заробітна плата, підпорядкованість тощо).

Розробити зручний графічний інтерфейс для роботи з інформацією. Основні дії та взаємодія між користувачем та системою повинні супроводжуватися відповідними повідомленнями для користувача.

Програма «Система підтримки прийняття рішень в управлінні персоналом» повинна бути універсальною, такою, щоб підходила як для домашнього користування, так і для підприємств.

Опис вхідної інформації

Подання вхідної інформації здійснюється через консоль. При додаванні та редагуванні працівника, наприклад, потрібно ввести інформацію про нього. Вхідна інформація проходить валідацію на правильність після введення.

Опис вихідної інформації

Дані про працівників зберігаються в словник. Вихідна інформація також представлена в консолі. Можливе виведення списку наявних працівників. Форма подання інформації представлена у вигляді списку з таких стрічок: «Id <ідентифікатор працівника>: <Ім’я>, <Посада>, <Зарплата >, Керівник: <Ім’я керівника>».

Опис алгоритму розв’язання задачі

З запуском програми в консоль виводиться інформація про можливі дії користувача, а саме: додати, вивести персонал, редагувати, видалити. Ці команди дозволяють користувачу взаємодіяти з системою прийняття рішень.

Додати – надає можливість користувачу вказати інформацію про працівника (ім’я, посада, зарплата, керівник) та додати його в систему. Після введення інформації про працівника, система створює об’єкт, присвоює його ідентифікатор та зберігає його у словник.

Редагувати – ­­­­­­­­­­­­­­­­­­­­­надає можливість користувачу змінити існуючу інформацію про працівника. Користувач може змінити як всі параметри працівника, так і окремі, залишаючи значення пустим.

Видалити – ­­­­­­­­­­­­­­­­­­­­­надає можливість користувачу видалити працівника з системи вказавши його ідентифікатор.

Вивести рперсонал – виводить список працівників в консоль. Система проходить через всіх зареєстрованих працівників і формує стрічку з їх даних у вигляді «Id <ідентифікатор працівника>: <Ім’я>, <Посада >, <Зарплата >, <Керівник>».

**РОЗДІЛ 2. РОЗРОБКА ЗАГАЛЬНОЇ СТРУКТУРИ ПРОГРАМИ**

* 1. **Аналіз функцій системи**

Програмний продукт оперує даними про працівників.

Завдання системи.

Система повинна застосовуватися для отримання інформації про працівників, додавання нових, редагування та видалення існуючих даних.

Система повинна виконувати основні функції:

* Реалізувати можливість додавання працівників, пошук, виведення наявного персоналу.
* Передбачити можливість зміни та видалення даних про працівників.
* Реалізувати ієрархічну структуру (одні працівники можуть підпорядковуватися іншим).
* Передбачити можливість фільтрування за різними показниками (функціональні обов’язки, заробітна плата, підпорядкованість тощо).

Не функціональні вимоги:

* для звичайної роботи програми достатньо комп’ютера з монітором клавіатурою та мишкою;
* простий і зрозумілий інтерфейс.
  1. **Розроблення структурної моделі системи**

Модель — опис об'єкта на якій-небудь формалізованій мові, складений з метою вивчення його властивостей. Такий опис особливо корисний у випадках, коли дослідження самого об'єкта ускладнене або фізично неможливе.

Натурне моделювання передує запуску масштабного промисловому випуску нової продукції. Часто в ролі моделі виступає інший матеріальний або уявний об'єкт, що замінює в процесі дослідження об'єкт-оригінал. Процес побудови моделі називається [моделюванням](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8E%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F).

Таким чином, модель виступає як своєрідний інструмент для пізнання, який дослідник ставить між собою і об'єктом, і за допомогою якого вивчає об'єкт, що його цікавить.

Моделі звичайно застосовуються для потреб [пізнання](https://uk.wikipedia.org/wiki/%D0%9F%D1%96%D0%B7%D0%BD%D0%B0%D0%BD%D0%BD%D1%8F_(%D1%84%D1%96%D0%BB%D0%BE%D1%81%D0%BE%D1%84%D1%96%D1%8F)) ([споглядання](https://uk.wikipedia.org/wiki/%D0%A1%D0%BF%D0%BE%D0%B3%D0%BB%D1%8F%D0%B4%D0%B0%D0%BD%D0%BD%D1%8F), [аналіза](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D1%96%D0%B7) і [синтеза](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B5%D0%B7)) і [конструювання](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D1%82%D1%80%D1%83%D1%8E%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F). Як модель може виступати [відображення](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%B4%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F), [схема](https://uk.wikipedia.org/wiki/%D0%A1%D1%85%D0%B5%D0%BC%D0%B0), [копія](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BF%D1%96%D1%8F), [макет](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D0%BA%D0%B5%D1%82), [зображення](https://uk.wikipedia.org/wiki/%D0%97%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F).

Моделлю може бути серійний повторюваний проєкт, що має набір певних, властивих тільки даної моделі параметрів і характеристик. Це робиться навіть в одному ряду виробів (проєктів). Модель рішень може мати кілька версій або варіантів, що є моделюванням діяльності, проєктування, управління великими проєктами тощо.

Процес створення моделі називається моделюванням.

Структурну модель системи ще називають структурною схемою. На структурній схемі відбивається склад системи та її внутрішні зв’язки. Для відображення структурної схеми системи використовуються графи.

Граф складається з вершин, що позначають елементи системи, і ребер – ліній, що позначають зв’язку (відносини) між елементами системи.

Структурна модель системи повинна передбачати роботу з користувацькими запитами (див. рис. 2.1).



*Рис. 2.1. Структурна модель системи*

Пояснення до рисунка:

* користувач може робити запити до системи, а саме «додати», «редагувати», «видалити», «вивести персонал» тощо;
* система обробляє запити користувача та взаємодіє зі словником працівників. Вона може записувати в словник нових працівників, діставати існуючих, редагувати та видаляти дані працівників. Також може надавати інформацію користувачу;
* словник зберігає працівників та дозволяє зчитувати та записувати їх.

**РОЗДІЛ 3. РОЗРОБЛЕННЯ ІЄРАРХІЇ КЛАСІВ**

* 1. **Декомпозиція системи**

Декомпозиція — науковий метод, що використовується структуру завдання і дозволяє змінити вирішення одного великого завдання рішенням серії менших завдань, нехай і взаємопов’язаних, але більш простих. Декомпозиція, як процес розділення, дозволяє розглядати будь-яку досліджувану систему як складну, що складається з окремих взаємопов'язаних підсистем, які, в свою чергу, також можуть бути розділеними на частини.

Метою декомпозиції є отримання вичерпного переліку класів, які необхідно реалізувати для побудови інформаційної системи з дотриманням принципів об’єктно-орієнтованого програмування.

Під час проведення декомпозиції слід дотримуватися принципів SOLID. Принципи SOLID використовуються для дизайну та розробки програмних систем, які, з великою ймовірністю, зможуть тривалий час розвиватися, розширюватися та підтримуватися. Дана концепція включає:

* принцип єдиного обов'язку (Single responsibility principle);
* принцип відкритості/закритості (Open/closed principle);
* принцип підстановки Лісков (Liskov substitution principle);
* принцип розділення інтерфейсу (Interface segregation principle);
* принцип інверсії залежностей (Dependency inversion principle).

Декомпозиція системи:

Клас Employee – представляє працівника, містить інформацію про нього та методи для роботи з ним.

Клас ControlSystem – представляє систему, яка буде виконувати основні функції системи, такі як створення, зміну, видалення працівників. Безпосередньо буде використовувати клас Employee.

* 1. **Аналіз взаємозв’язків між об’єктами**

Структура класу Employee:

first: str – ім’я працінивка;

last: str – прізвище працівника;

position: str – посада працівника;

pay: int – зарплата працівника;

email: str – електронна скринька працівника;

manager: Employee – керівник;

Структура класу ControlSystem:

employees: dict – словник, в якому зберігаються працівники.

* 1. **Розроблення інтерфейсів класів**

Інтерфейс класу Employee:

fullname – формує повне ім’я працівника;

get\_manager – повертає керівника;

set\_manager – задає керівника працівнику;

from\_string: classmethod – створює об’єкт класу зі стрічки.

Інтерфейс класу ControlSystem:

input\_employee –просить ввести дані працівника в користувача;

input\_manager – просит ввести дані про керівника в користувача;

add\_employee – додає працівника в систему;

find\_employee – знаходить працівника по ідентифікатору;

edit\_employee – змінює дані працівника;

delete\_employee – видаляє працівника з системи;

show\_personnel – виводить інформацію про персонал.

* 1. **UML-діаграма класів**

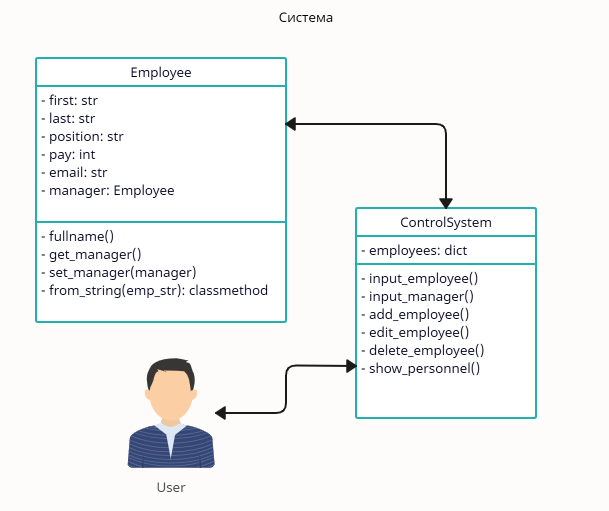
Unified Modeling Language (UML) — уніфікована мова моделювання. Modeling передбачає створення моделі, що описує об'єкт. Unified (універсальний, єдиний) — підходить для широкого класу проектованих програмних систем, різних областей додатків, типів організацій, рівнів компетентності, розмірів проектів. UML описує об'єкт в єдиному заданому синтаксисі, тому де б ви не намалювали діаграму, її правила будуть зрозумілими для всіх, хто знайомий з цією графічною мовою.

Як будь-яка інша мова, UML має власні правила оформлення моделей і синтаксис. За допомогою графічної нотації UML можна візуалізувати систему, об'єднати всі компоненти в єдину структуру, уточнювати і покращувати модель у процесі роботи.

Діаграма класів — статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як класи, типи, їх зміст та відношення. Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

Класс (class) — категорія речей, що мають загальні атрибути та операції. Сама діаграма класів являє собою набір статичних, декларативних елементів моделі. Вона дає нам найбільш повне і розгорнуте уявлення про зв'язки в програмному коді, функціональність та інформацію про окремі класи.

Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини (див. рис. 3.1).

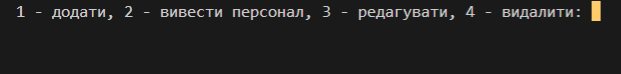


*Рис. 3.1. UML-діаграма системи*

**РОЗДІЛ 4. РОЗРОБКА СИСТЕМИ ВВОДУ-ВИВОДУ ДАНИХ**

Принципом взаємодії користувача з розробленим програмним забезпеченням являється запит від користувача та надання необхідних даним користувачем при потребі.

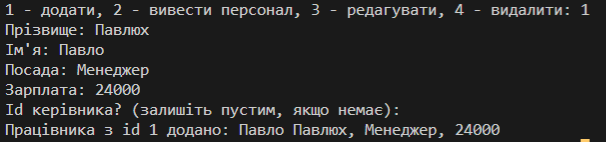
При запуску програми система пропонує вибрати можливу дію користувачу (див. рис. 4.1).



*Рис. 4.1. Початковий інтерфейс програми*

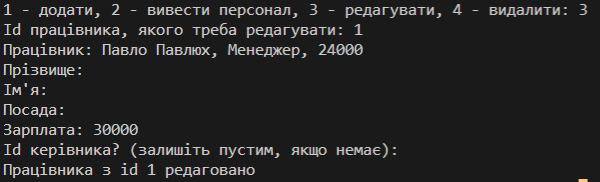
Користувач може вибрати один з наведених дій зі списку, а саме «додати», «вивести персонал», «редагувати» і «видалити». Кожна з цих дій повинна бути прописана відповідною цифрою користувачем в консолі для її виклику.

Додати ­­­­­­­­­­– просить надати користувача необхідні дані для створення працівника, створює та додає його до системи після перевірки введених даних. Виводить ідентифікатор та інформацію про працівника після додавання його до системи (див. рис. 4.2).



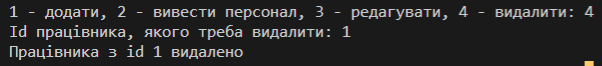
*Рис. 4.2. Вигляд дії «додати»*

Редагувати – вимагає ввести ідентифікатор працівника, для його редагування. Якщо такий ідентифікатор існує – просить ввести нові дані працівника, або залишити пустим для використання попередніх даних (див. рис. 4.3).



*Рис. 4.3. Вигляд дії «редагувати»*

Видалити – просить ввести ідентифікатор працівника, для його видалення з системи. Виводить результат видалення (див. рис. 4.4).



*Рис. 4.4. Вигляд дії «видалити»*

Вивести персонал – виводить всіх працівників, які зареєстровані в системі (див. рис. 4.5).



*Рис. 4.5. Вигляд дії «вивести персонал»*

**РОЗДІЛ 5. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

Тестування програмного забезпечення - процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, здійснюваний шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином.

Може оцінюватись:

* відповідність вимогам, якими керувалися проектувальники та розробники;
* правильна відповідь для усіх можливих вхідних даних;
* виконання функцій за прийнятний час;
* практичність;
* сумісність з програмним забезпеченням та операційними системами;
* відповідність задачам замовника.

Оскільки число можливих тестів навіть для нескладних програмних компонент практично нескінченне, тому стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів. Як результат програмне забезпечення тестується стандартним виконанням програми з метою виявлення помилок або інших дефектів.

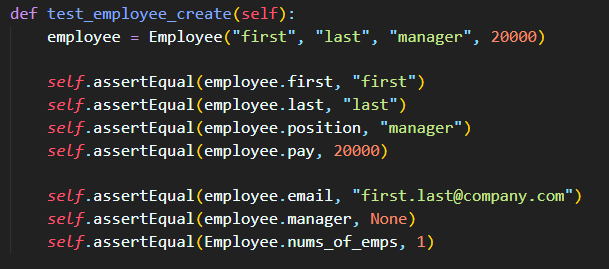
Тестування ПЗ може надавати об'єктивну, незалежну інформацію про якість ПЗ, ризики відмови, як для користувачів, так і для замовників.

Тестування може проводитись, як тільки створено виконуваний код (навіть частково завершений). Процес розробки зазвичай передбачає, коли та як буде відбуватися тестування. Наприклад, при поетапному процесі, більшість тестів відбувається після визначення системних вимог і тоді вони реалізуються в тестових програмах. На противагу цьому, відповідно до вимог гнучкої розробки ПЗ, програмування і тестування часто відбувається одночасно.

Тестування програми «Система підтримки прийняття рішень в управлінні персоналом»:

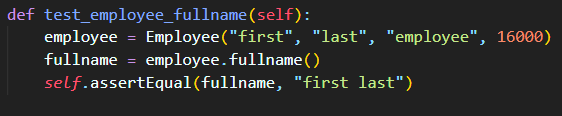
Employee Test Case: містить набір тестів, які перевіряють коректність створення працівників та методів класу Employee.

* Тест test\_employee\_create: перевіряєтворення об’єкту працівника та його даних (див. рис. 5.1).



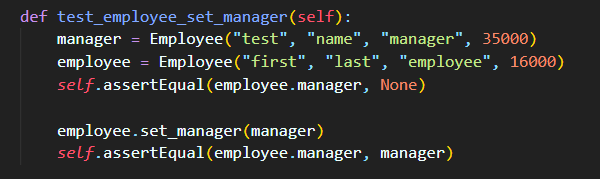
*Рис. 5.1.*

* Тест test\_employee\_fullname: тестує правильність формування повного імені працівника (див. рис. 5.2).



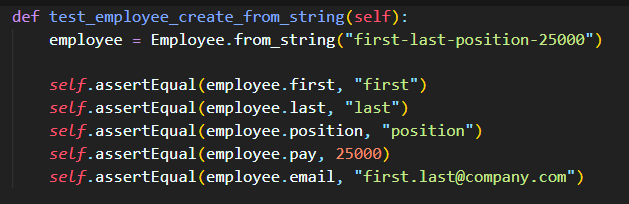
*Рис. 5.2.*

* Тест test\_employee\_set\_manager: перевіряє роботу методу задання керівника працівнику (див. рис. 5.3).



*Рис. 5.3.*

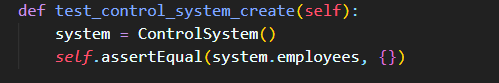
* Тест test\_employee\_create\_from\_string: тестує створення працівника зі стрічки (див. рис. 5.4).



*Рис. 5.4.*

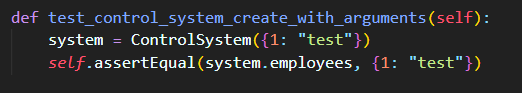
Control System Test Case: перевіряє роботу методів класу ControlSystem та його атрибутів, тим самим перевіряється коректність роботи системи.

* Тест test\_control\_system\_create: тестує створення системи та її атрибутів (див. рис. 5.5).



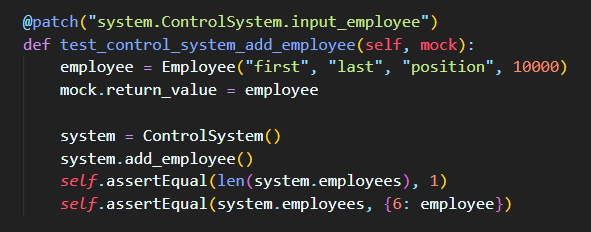
*Рис. 5.5.*

* Тест test\_control\_system\_create\_with\_arguments: перевіряє створення системи з вхідним параметром (див. рис. 5.6).



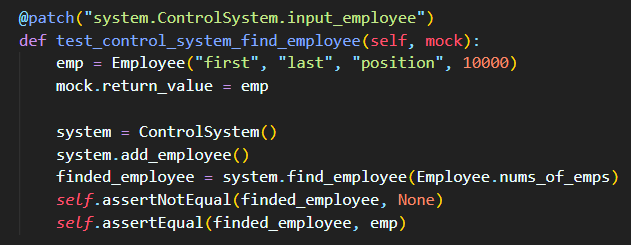
*Рис. 5.6.*

* Тест test\_control\_system\_add\_employee: тестує додавання працівника у систему (див. рис. 5.7).



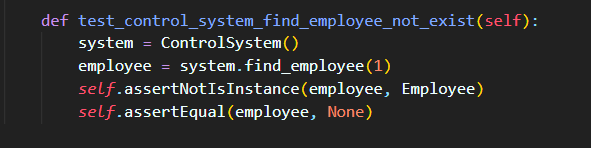
*Рис. 5.7.*

* Тест test\_control\_system\_find\_employee: перевіряє пошук існуючого працівника в системі (див. рис. 5.8).



*Рис. 5.8.*

* Тест test\_control\_system\_find\_employee\_not\_exist: тестує пошук неіснуючого працівника в системі (див. рис. 5.9).



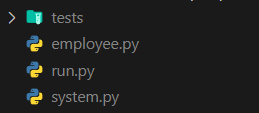
*Рис. 5.9.*

**РОЗДІЛ 6. РОЗРОБКА ДОКУМЕНТАЦІЇ ДЛЯ СУПРОВОДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ**

Документація до проекту:

Опис стеку технологій: проект виконаний на мові програмування Python в середовищі Visual Studio Code.

Файлова структура проекту: проект розділений на модулі. Це обумовлюється зручністю використання, швидкістю знаходження помилок та легкою заміною модуля на інший при потребі. Структура містить такі модулі: папку tests, в якій знаходить набір тест кейсів, employee.py, в якому описана логіка класу працівника, system.py, де знаходиться код системи та run.py, для запуску програми (див. рис. 6.1).



*Рис. 6.1. Структура файлової системи*

Запуск проекту:

Для стабільної роботи рекомендується встановити віртуальне оточення Python. Це можливо зробити за допомогою команди «python -m venv <назва папки оточення>». Після виконання команди створиться папка з вказаною назвою. Віртуальне оточення потрібно активувати, за допомогою команди «<назва папки оточення>\Scripts\activate>» (Windows), або «source <назва папки оточення >/bin/activate» (MacOS, Linux).

Запуск виконається за допомогою команди «python run.py» в консолі. Для запуску тестів потрібно ввести команду «python -m unittest discover tests -b».

Вимоги до системи:

Проект можливо запустити на любій сучасній операційній системі, такій як Windows, MacOS, Linux, з встановленим інтерпретатором Python. Версія Python повинна відповідати 3.7, або вище, для коректної роботи проекту.

**ВИСНОВКИ**

В ході виконання роботи мовою Python в середовищі Visual Studio Code реалізовано програму «Система підтримки прийняття рішень в управлінні персоналом». Також продемонстровано на прикладі можливості об’єктно-орієнтованого програмування.

Реалізовано такі функціональні можливості:

* Можливість додавання працівника в систему;
* Можливість пошуку працівника по ідентифікатору;
* Можливість редагування даних працівника;
* Можливість видалення працівника з системи;
* Можливість виводу персоналу;
* Можливість задання керівника працівнику.

Об’єктно-орієнтоване програмування дозволяє створювати програму, як набір користувацьких типів даних (класів), приховувати деталі реалізації, використовувати повторний код, інтерпретувати виклики процедур та функцій на етапі виконання (втілюючи основи ООП - інкапсулювання, поліморфізм, успадкування).

На закінчення терміну на виконання курсової роботи залишились не реалізованими такі функціональні вимоги: можливість фільтрування працівників за різними показниками.

Універсальність програми полягає в тому, що її інтерфейс простий та зрозумілий, кожен користувач може створити свою систему працівників.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. **John M. Zelle «Python Programming: An Introduction to Computer Science».**
2. **Ден Бейдер «Чистий Python. Тонкощі програмування для профі».**
3. **Ел Свейгарт «Автоматизація рутинних задач за допомогою Python. Практиче керівництво для початківців».**
4. https://app.creately.com/ - Побудова UML-діаграми.
5. <https://studfile.net/> - Теоретична інформація.