**Зміст**

[Вступ 3](#_Toc71405242)

[1. АНАЛІЗ СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАВДАННЯ.](#_Toc71405243) 4

[1.1 Аналіз предмету проектування](#_Toc71405244) 4

[1.2 Класифікація об’єктно-орієнтованих мов програмування 7](#_Toc71405245)

[1.3 Огляд та аналіз сучасних технологій та засобів проектування програмного забезпечення 11](#_Toc71405246)

[1.4 Універсальна мова проектування UML 12](#_Toc71405247)

[1.5 Уточнена постановка задачі та розробку програмного забезпечення](#_Toc71405248) 13

[2. РОЗРОБКА ЗАГАЛЬНОЇ СТРУКТУРИ ПРОГРАМИ.](#_Toc71405249) 14

[2.1 Аналіз функцій системи.](#_Toc71405250) 14

[2.2 Розроблення структурної моделі системи.](#_Toc71405251) 14

[3. РОЗРОБЛЕННЯ ІЄРАРХІЇ КЛАСІВ.](#_Toc71405252) 17

[3.1 Декомпозиція системи.](#_Toc71405253) 17

[3.2 Аналіз взаємозв’язків між об’єктами.](#_Toc71405254) 21

[3.3 Розроблення інтерфейсів класів.](#_Toc71405255) 25

[3.4 UML-діаграма класів.](#_Toc71405256) 26

[4. РОЗРОБКА СИСТЕМИ ВВОДУ-ВИВОДУ ДАНИХ.](#_Toc71405257) 28

[5. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ](#_Toc71405258) 30

[6. РОЗРОБКА ДОКУМЕНТАЦІЇ ДЛЯ СУПРОВОДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ](#_Toc71405259) 32

[ВИСНОВОК](#_Toc71405260) 37

[БІБЛІОГРАФІЧНИЙ СПИСОК](#_Toc71405261) 38

[ДОДАТОК](#_Toc71405262) 39

## Вступ

Об'єктно-орієнтоване програмування (ООП) — одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають чотири основні концепції: інкапсуляція, успадкування, поліморфізм та абстракція. Однією з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця парадигма з'явилась в 1960-х роках, вона не мала широкого застосування до 1990-х, коли розвиток комп'ютерів та комп'ютерних мереж дав змогу писати надзвичайно об'ємне і складне програмне забезпечення, що змусило переглянути підходи до написання програм.

На відміну від традиційних поглядів, коли програму розглядали як набір підпрограм, або як перелік інструкцій комп'ютеру, ООП-програми можна вважати сукупністю об'єктів. Відповідно до парадигми об'єктно-орієнтованого програмування, кожен об'єкт здатний отримувати повідомлення, обробляти дані, та надсилати повідомлення іншим об'єктам. Кожен об'єкт — своєрідний незалежний автомат з окремим призначенням та відповідальністю.

В результаті дослідження Дебори Дж. Армстронг комп'ютерної літератури, що була видана протягом останніх 40 років, вдалось відокремити фундаментальні поняття (принципи), використані у переважній більшості визначень об'єктно-орієнтованого програмування. До них належить:

Клас: визначає абстрактні характеристики деякої сутності, включно з характеристиками самої сутності (її атрибутами або властивостями) та діями, які вона здатна виконувати (її поведінкою, методами або можливостями). Наприклад, клас Собака може характеризуватись рисами, притаманними всім собакам, зокрема: порода, колір хутра, здатність гавкати. Класи вносять модульність та структурованість в об'єктно-орієнтовану програму.

Об'єкт: Окремий екземпляр класу (створюється після запуску програми і ініціалізації полів класу). Клас Собака відповідає всім собакам шляхом опису їхніх спільних рис; об'єкт Сірко є одним окремим собакою, окремим варіантом значень характеристик. Собака має хутро; Сірко має коричнево-біле хутро. Об'єкт Сірко є екземпляром (примірником) класу Собака. Сукупність значень атрибутів окремого об'єкта називається станом. На основі класу Собака можна, також, створити інший об'єкт Дружок, який відрізнятиметься від об'єкта Сірко своїм станом (наприклад кольором хутра). Обидва об'єкта (Сірко і Дружок) є екземплярами класу Собака.

Метод: Можливості об'єкта. Оскільки Сірко — Собака, він може гавкати. Тому гавкати() є одним із методів об'єкта Сірко. Він може мати й інші методи, зокрема: місце(), або їсти(). В межах програми, використання методу має впливати лише на один об'єкт; всі Собаки можуть гавкати, але треба щоб гавкав лише один окремий собака.

Обмін повідомленнями :«Передача даних від одного процесу іншому, або надсилання викликів методів.»

Успадкування (наслідування): Клас може мати «підкласи», спеціалізовані, розширені версії надкласу. Можуть навіть утворюватись цілі дерева успадкування. Наприклад, клас Собака може мати підкласи Коллі, Пекінес, Вівчарка тощо. Так, Сірко може бути екземпляром класу Вівчарка. Підкласи успадковують атрибути та поведінку своїх батьківських класів, і можуть вводити свої власні. Успадкування може бути одиничне (один безпосередній батьківський клас) та множинне (кілька батьківських класів). Це залежить від вибору програміста, який реалізовує клас та мови програмування. Так, наприклад, в Java дозволене лише одинарне успадкування, а в С++ і те і інше.

Приховування інформації (інкапсуляція): Приховування деталей про роботу класів від об'єктів, що їх використовують або надсилають їм повідомлення. Так, наприклад, клас Собака має метод гавкати(). Реалізація цього методу описує як саме повинно відбуватись гавкання (приміром, спочатку вдихнути() а потім видихнути() на обраній частоті та гучності). Петро, хазяїн пса Сірка, не повинен знати як він гавкає. Інкапсуляція досягається шляхом вказування, які класи можуть звертатися до членів об'єкта. Як наслідок, кожен об'єкт надає кожному іншому класу певний інтерфейс — члени, доступні іншим класам. Інкапсуляція потрібна для того, аби запобігти використанню користувачами інтерфейсу тих частин реалізації, які, швидше за все, будуть змінюватись. Це дасть змогу полегшити внесення змін без потреби змінювати і користувачів інтерфейсу. Наприклад, інтерфейс може гарантувати, що щенята можуть додаватись лише до об'єктів класу Собака кодом самого класу. Часто, члени класу позначаються як публічні (англ. public), захищені (англ. protected) та приватні (англ. private), визначаючи, чи доступні вони всім класам, підкласам, або лише до класу в якому їх визначено.

Абстрагування: Спрощення складної дійсності шляхом моделювання класів, що відповідають проблемі, та використання найприйнятнішого рівня деталізації окремих аспектів проблеми. Наприклад Собака Сірко більшу частину часу може розглядатись як Собака, а коли потрібно отримати доступ до інформації специфічної для собак породи коллі — як Коллі і як Тварина (можливо, батьківський клас Собака) під час підрахунку тварин Петра.

Поліморфізм: Поліморфізм означає залежність поведінки від класу, в якому ця поведінка викликається, тобто, два або більше класів можуть реагувати по-різному на однакові повідомлення. Наприклад, якщо Собака отримує команду голос(), то у відповідь можна отримати Гав; якщо Свиня отримує команду голос (), то у відповідь можна отримати Рох-рох. На практиці - це реалізовується шляхом реалізації ряду підпрограм (функцій, процедур, методи тощо) з однаковими іменами, але з різними параметрами. Залежно від того, що передається, і вибирається відповідна підпрограма.

1. **АНАЛІЗ СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАВДАННЯ.**
   1. **Аналіз предмету проектування**

Основна проблема сучасного Інтернету - надлишок інформації, яку людина не в змозі систематизувати вручну. Дуже часто така річ як «Парсинг» спрощує роботу людей. Парсинг можна використовувати для того щоб зрозуміти середню вартість тих чи інших товарів на ринку, зручно використовувати дані по конкурентах. Однак якщо це сотні і тисячі позицій, зібрати їх вручну оперативно неможливо. Парсинг можна здійснювати на регулярній основі, наприклад, щотижня, виявляючи на що підвищилися ціни в середньому по ринку і які новинки з'явилися у конкурентів. Частим явище являється наведення порядку на власному веб-ресурсі, за допомогою парсингу стає можливим знайти неіснуючі сторінки, дублі, неповний опис, відсутність певних характеристик або невідповідність даних по складських залишків того, що відображається на сайті. З парсером швидше. Існує парсинг, пов'язаний зі складанням, наприклад, списку осіб, які приймають рішення, в тій чи іншій галузі і місті. Для цього може застосовуватися особистий кабінет на сайтах пошуку роботи з доступом до актуальних і архівних резюме. Етичність подальшого використання подібної бази кожна компанія визначає самостійно.

За допомогою стає можливим створення програми яка не буде постійно збирати, аналізувати та виводи інформацію з інтернет ресурсу, при цьому дані не будуть статичні оскільки беруться з інтернет ресурсу який має свою власну базу даних. Можна навіть збирати різну інформацію з декількох сайтів. Саме таке завдання буде виконуватись у цій курсовій роботі.

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

Три основні класифікації склалися історично:

1. За функціональною силою:

1) універсальні мови ( в них можна промоделювати, умовно кажучи, будь-який алгоритм);

2) спеціалізовані мови (орієнтовані на певні класи задач).

2. 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) і т. п.

3. За рівнем абстракції:

1) мови низького рівня (машинно-залежні) - Assembler і т. п.;

2) мови високого рівня (орієнтовані на користувача (людину) до певної міри) - Pascal, C, Fortran і т. п.;

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 й обладнанням, 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.

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.

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.

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

**Python**

Інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.

Дизайн мови Python побудований навколо об'єктно-орієнтованої моделі програмування. Реалізація ООП в Python є елегантною, потужною та добре продуманою, але разом з тим, достатньо специфічною в порівнянні з іншими об'єктно-орієнтованими мовами.

Можливості та особливості:

* Класи є одночасно об'єктами з усіма нижче наведеними можливостями.
* Успадкування, в тому числі множинне.
* Поліморфізм (всі функції віртуальні).
* Інкапсуляція (два рівні — загальнодоступні та приховані методи і поля). Особливість — приховані члени доступні для використання та помічені як приховані лише особливими іменами.
* Спеціальні методи, що керують життєвим циклом об'єкта: конструктори, деструктори, розподільники пам'яті.
* Перевантаження операторів (усіх, крім is, '.', '=' і символьних логічних).
* Властивості (імітація поля за допомогою функцій).
* Управління доступу до полів (емуляція полів і методів, частковий доступ тощо).
* Методи для управління найпоширенішими операціями (істиннісне значення, len(), глибоке копіювання, серіалізація, ітерація по об'єкту, …)
* Метапрограмування (управління створенням класів, тригери на створення класів, та ін)
* Повна інтроспекція.
* Класові та статичні методи, класові поля.
* Класи, вкладені у функції та інші класи.
  1. **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.

* 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 Object Management Group (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 та розробку програмного забезпечення**

На основі виконаного аналізу та огляду літературних джерел можна сформулювати постановку задачі: розробити "Агрегатор метеорологічних даних", основними задачами якого є: можливість отримання інформацію о погоді з двох метеорологічних ресурсів:

1. Sinoptik.ua
2. Meteoprog.ua

Інформація має виводитись при вводі користувачем назви населеного пункту а також виводити точність прогнозу для вищезгаданих джерел.

1. **РОЗРОБКА ЗАГАЛЬНОЇ СТРУКТУРИ ПРОГРАМИ.**
   1. **Аналіз функцій системи.**

Як було позначено а главі **1.5** «**Уточнена постановка задачі та розробку програмного забезпечення»** агрегатор повинен виводити інформацію про погоду у конкретному населеному пункті. Програма буде мати можливість «Парсити» інформацію з вищеперечислених джерел.

Парсинг (від англ. Parse) – процес аналізу або розбору певного контенту на складові за допомогою роботів-парсерів (спеціальних програм або скриптів). У SEO цим контентом є html-код сторінок сайтів.

Часто парсинг плутають з граббінгом. Це близькі поняття, але все ж мають різні значення. Граббер дозволяє скачувати інформацію з мережі (html-сторінки, rss-стрічки, xml-документи) в свою базу, а парсер дозволяє виявити з цієї купи корисну інформацію і обробити її, залежно від поставлених завдань.

Система повинна виконувати основні функції:

* Надання можливості користувачу введення назви населеного пункту
* Можливість виведення програмою інформації про дату
* Можливість виведення інформації про температуру повітря
* Можливість виведення точності прогнозу для джерел з яких парситься інформація
  1. **Розроблення структурної моделі системи.**

Програма буде мати п’ять класів кожен з яких буде мати свої змінні та методи які будуть з ними взаємодіяти. Поділимо програму з класами на кілька структурних блоків, кожен клас буде відповідати за свій блок і один блок для бібліотек у самого початку коду:

Блок перший: Використані бібліотеки

* Бібліотека яка має створювати URL на який буде надіслано запит
* Бібліотека яка відповідає за запити на сервер для отримання інформації
* Бібліотека яка для маніпулювання інформацією отриманої із запиту

Блок другий: отримання URl

Блок має завдання створення та повернення

URL адресу для джерел на які буде надіслано

запит

Блок третій: надсилання запиту

Блок має завдання надсилання запиту за URL

адресою та отримання інформації як відповідь

від серверу

Блок четвертий: батьківіський клас

Блок обробляє отриману відповідь від сервера,

та переводить її у відповідний вид для виводу.

Також блок служить батьківським класом для

двох наступних блоків, вони відповідно наслідують

всі його властивості та методи

Блок п'ятий (bro): вивід інформації

Блок отримує інформацію від батьківського класу,

обробляє її та виводить

Блок шостий (bro): вивід інформації

Блок отримує інформацію від батьківського класу,

обробляє її та виводить

1. **РОЗРОБЛЕННЯ ІЄРАРХІЇ КЛАСІВ.**
   1. **Декомпозиція системи.**

Декомпозиція — науковий метод, що використовує структуру завдання і дозволяє замінити 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. Як системи можуть виступати не тільки матеріальні об'єкти, а й процеси, явища і поняття. Спрощене графічне представлення декомпозиційованої системи називається її ієрархічною структурою. Ієрархічна структура може бути зображена у вигляді розгалуженої блок-схеми.

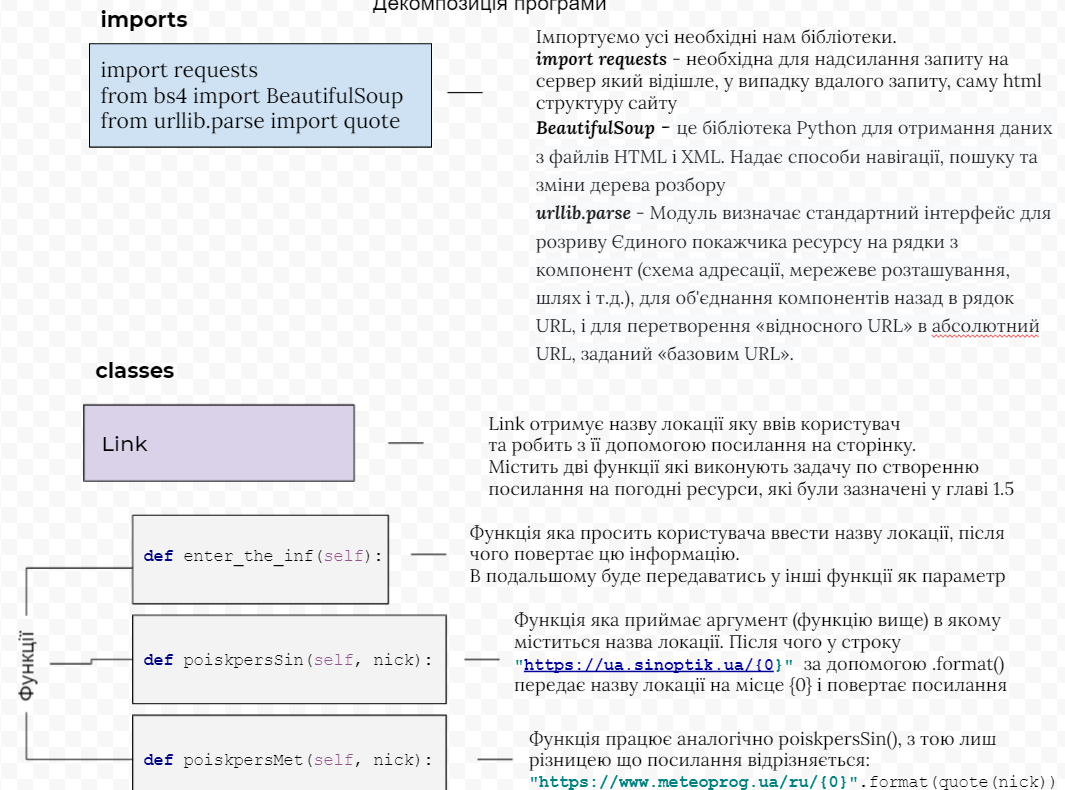


Рис. 3.1 Графічне представлення декомпозиційованої системи "Link”

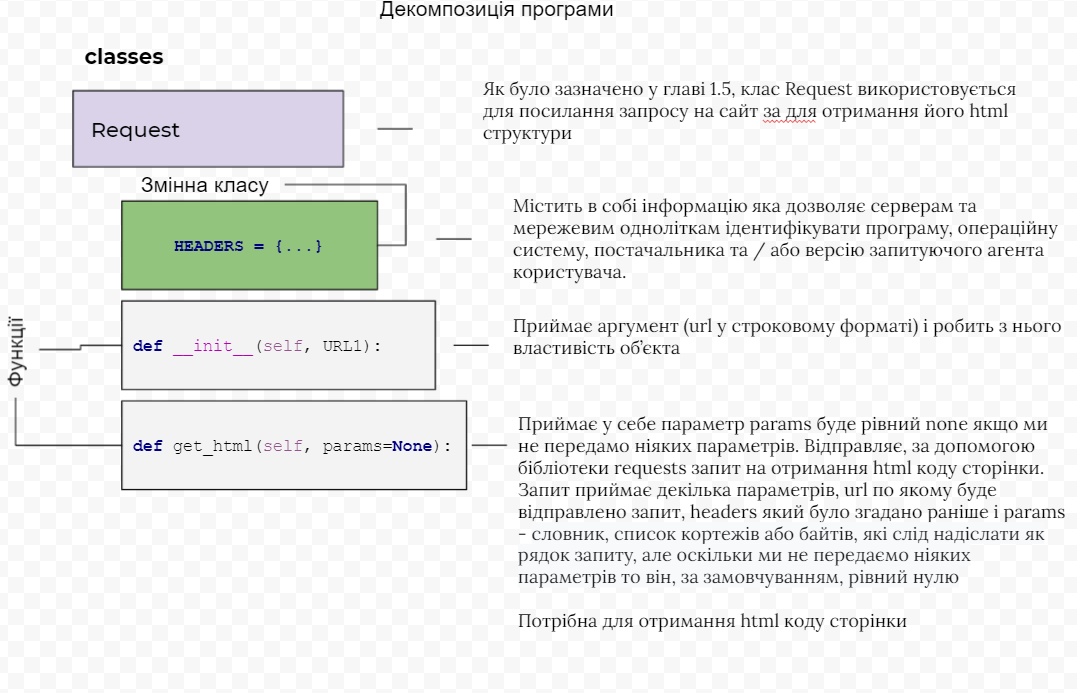
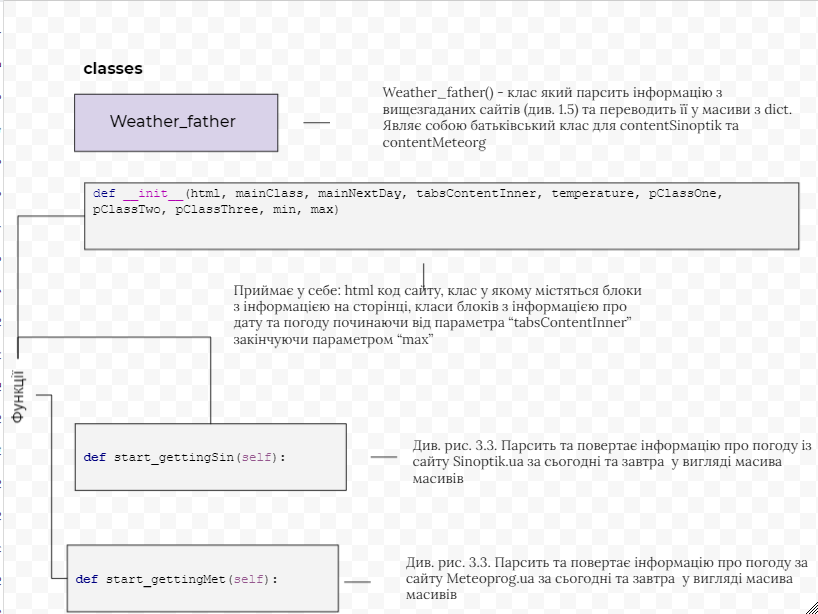


Рис. 3.2 Графічне представлення декомпозиційованої системи “Request”



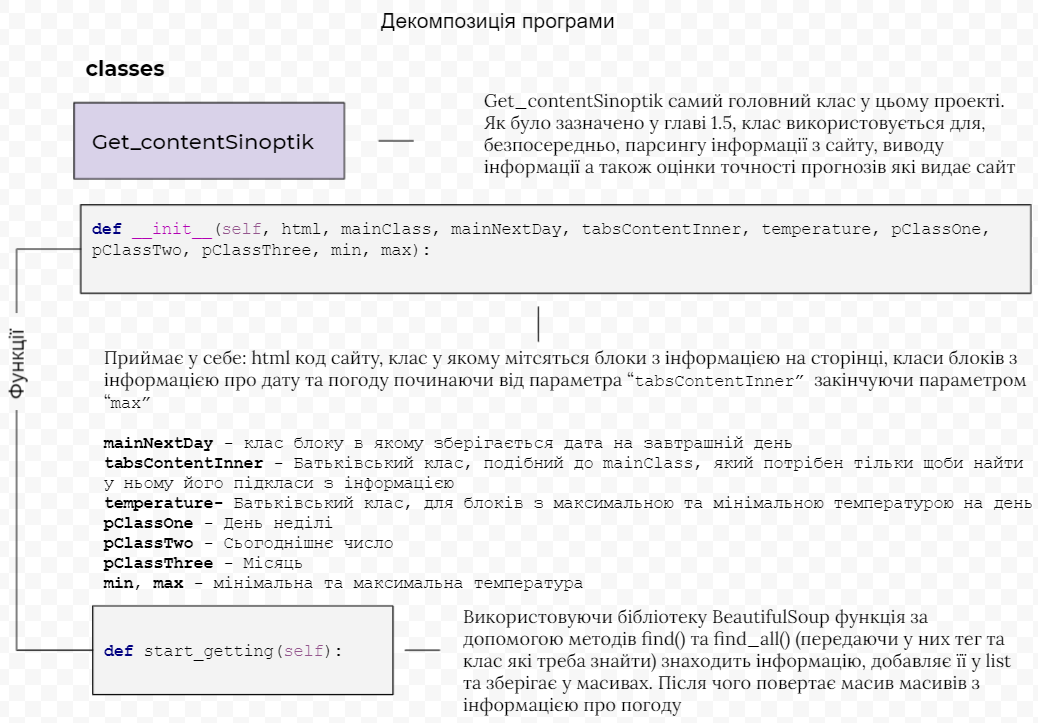


Рис. 3.3 Графічне представлення декомпозиційованої системи “Weather\_father”

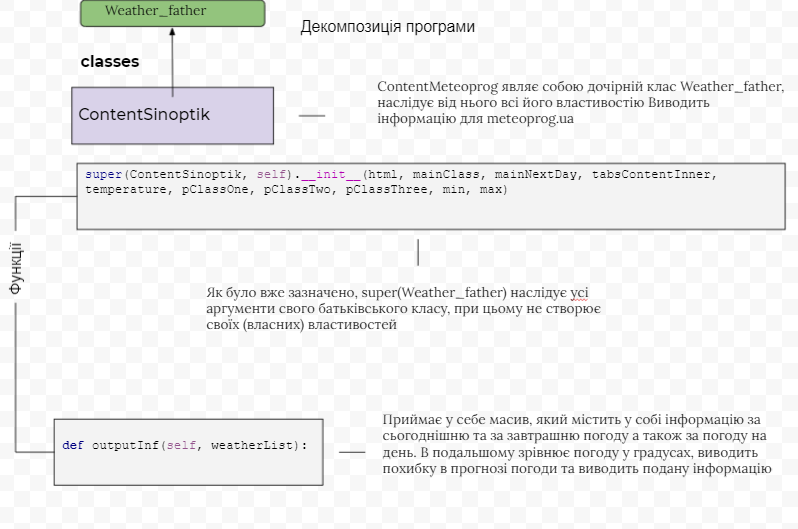


Рис. 3.4 Графічне представлення декомпозиційованої системи “ContentSinoptik”

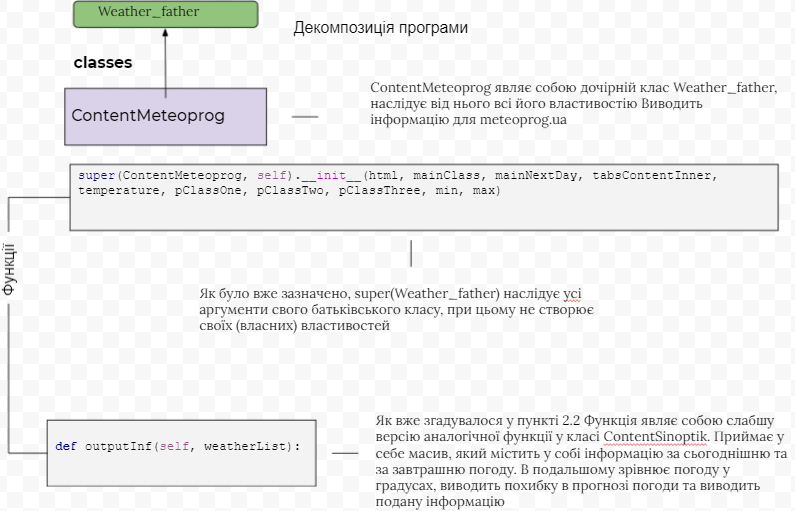


Рис. 3.5 Графічне представлення декомпозиційованої системи “ContentMeteoprog”

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

У об’єктно-орієнтованому програмуванні існує дві ієрархії: класів та об’єктів. За допомогою об’єктно-орієнтованого програмування є можливість описати різні об’єкти, їх характеристики та властивості. Об’єкти описуються за допомогою класів. Об’єкт це сутність зі своїми властивостями та характеристиками, яка може реагувати на повідомлення за допомогою своїх існуючих даних та з’являється при створення екземпляра класу або прототипу. Клас – це список характеристик та можливостей об’єкта.

У об’єктно-орієнтованому програмуванні існують таки зв’язки (відношення) між класами та об’єктами:

* асоціація;
* успадкування;
* агрегація;
* використання;
* інстанциювання;
* метаклас.

Успадкування класів – це дуже потужна можливість, яка дозволяє створювати похідні класи. Один клас може успадковувати властивості та характеристики іншого класу. При цьому клас який успадковує ці властивості називається дочірнім (похідним), а клас, який саме містить ці властивості – батьківським (базовим). Таким чином буде зекономлено маса часу на написання та налагодження коду нової програми. Об'єкти похідного класу вільно можуть використовувати все, що створено і налагоджено в базовому класі. При цьому, є можливість в похідний клас дописати необхідний код для удосконалення програми: додати нові елементи, методи тощо. Базовий клас залишиться недоторканим.

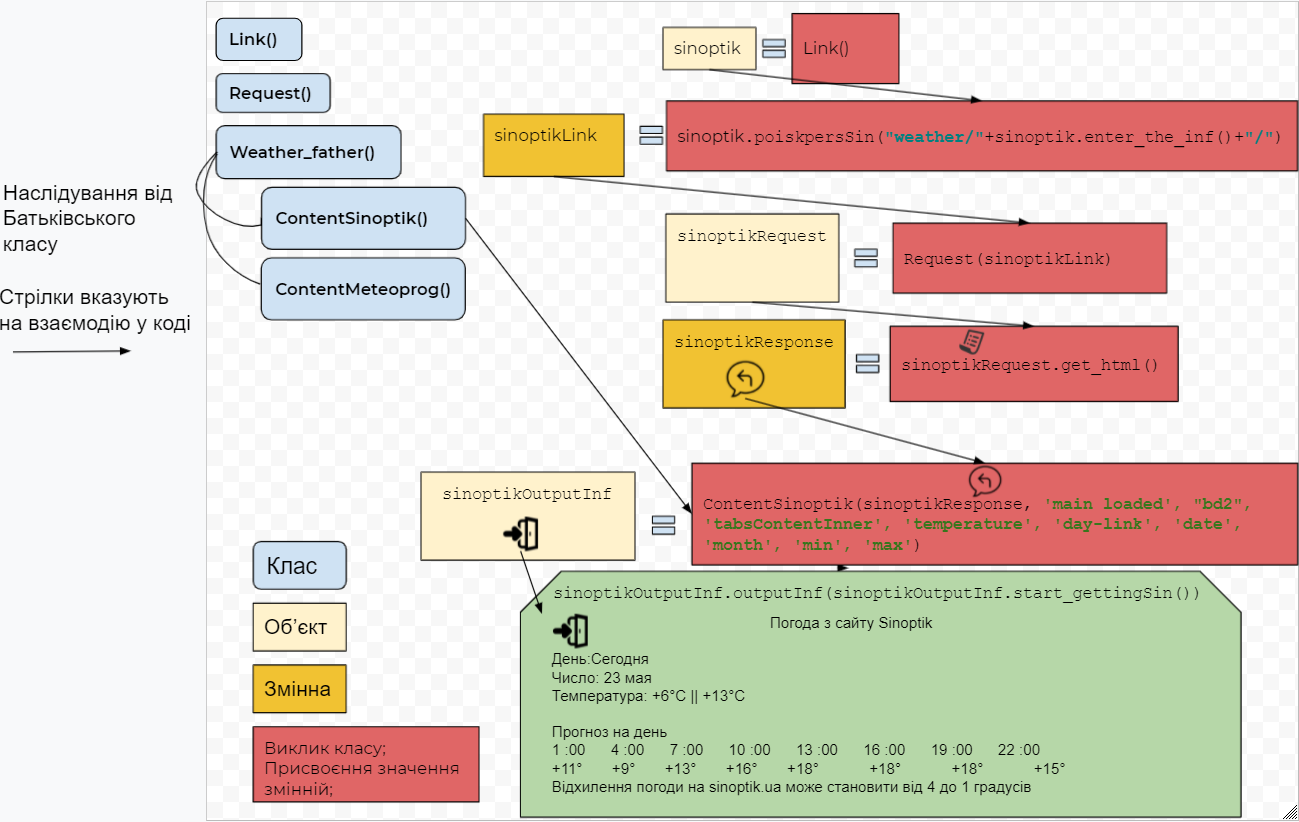


Рис. 3.6 Малюнок зв’язків між об’єктами та класами **sinoptik**

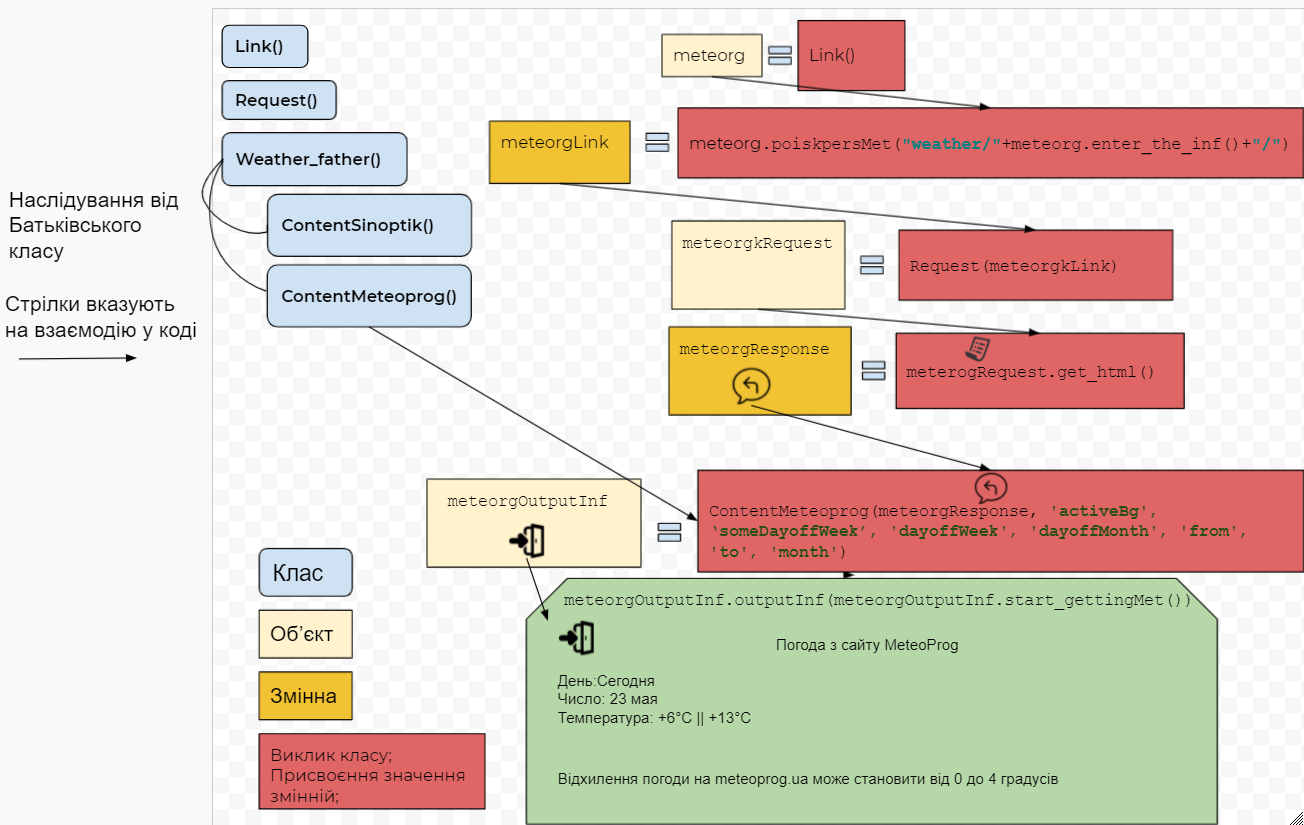


Рис. 3.7 Малюнок зв’язків між об’єктами та класами **meteoprog**

В проекті використовуються 5 класів: class Link(), Request(URL), Weather\_father(), ContentSinoptik(Weather\_father, Link), ContentMeteoprog(Weather\_father).

Клас Request () отримує інформацію із об’єкта, який створений на основі класа Link() для подальшого запросу на сервер. Далі об’єкту, створеному на основі класа Request(), передається як аргумент змінна meteorgLink яка в свою чергу повертає посилання на ресурс, далі використовуються метод для отримання html відповіді з ресурсу. На основі класу Weather\_father створюються братерські класи ContentSinoptik(), та ContentMeteoprog(). Як показано на рис. 3.5-3.6 створюються нові об’єкти на основі класів ContentSinoptik() та ContentMeteoprog(). Об’єкти, наслідуючись від батьківського класу, приймають необхідні аргументи, перетворюють їх у властивості. Об’єкти викликають функції start\_gettingSin() та start\_gettingMet(), оскільки цих функцій немає у створених об’єктах вони шукають їх у батьківському класі. Функція шукає інформацію по переданим класам які були передані, як властивості до об‘єктів та повертає її у вигляді масиву. Масив приймає викликана об’єктами функція outputInf(), обробляє її та виводить інформацію про погоду у консоль

* 1. **Розроблення інтерфейсів класів.**

Interface Link:

**{**  
 **def** \_\_int\_\_(self, none) **pass  
  
 def** enter\_the\_inf(self) -> x:str  **def** poiskpersSin(self, nick: str) -> geourl: str **def** poiskpersMet(self, nick: str) -> geourl: strgeourl: str  
 format(): function  
 quote(nick): function  
 **pass  
 }**Interface Request:

**{**  
 HEADERS: dict **def** \_\_init\_\_(self, URL1: str)  
  **def** get\_html(self, params=**None**) -> r.text: str

**}**

Interface Weather\_father():

**{**

**def** \_\_init\_\_(self, html:str, mainClass: str, mainNextDay: str, tabsContentInner: str, temperature: str, pClassOne: str, pClassTwo: str, pClassThree: str, min: str, max: str):

*"Приймає параметри для пошуку, див. рис. 3.3"* **def** start\_gettingSinoptik(self, none) -> list[3]  
 **return** [weather1, weather2, weather3, weatherNextDay]

**def** start\_gettingMeteoprog(self, none) -> list[1]  
 **return** [weather1, weather2]

**}**

Interface ContentSinoptik(Weather\_father: **class**, Link: **class**):  
 super(ContentSinoptik, self).\_\_init\_\_(html:str, mainClass:str, mainNextDay:str, tabsContentInner:str, temperature:str, pClassOne:str, pClassTwo:str, pClassThree:str, min:str, max:str)  
 **"Наслідує параметри для пошуку з батьківського класу Weather\_father"**

**{**  
   
 **def** outputInf(self, weatherList: list):  
 *"Функція отримує масив масивів та опрацьовує його"*

**}**

Interface ContentMeteoprog(Weather\_father: **class**):  
 super(ContentMeteoprog, self).\_\_init\_\_(html:str, mainClass:str, mainNextDay:str,  
tabsContentInner:str, temperature:str, pClassOne:str, pClassTwo:str, pClassThree:str, min:str, max:str)  
 **"Наслідує параметри для пошуку з батьківського класу Weather\_father"**

**{**  
  
 **def** outputInf(self, weatherList: list):

*"Функція отримує масив масивів та опрацьовує його"*  
 **}**

*# Вивід информації з sinoptik* sinoptik: object  
sinoptikLink: str  
sinoptikRequest: object  
sinoptikResponse: dict  
sinoptikOutputInf: object  
sinoptikOutputInf.outputInf(sinoptikOutputInf.start\_gettingSinoptik())

*# Вивід информації з meteoprog*meteorg: object  
meteorgLink: str  
meteorgRequest: object  
meteorgResponse: dict  
meteorgOutputInf: object  
meteorgOutputInf.outputInf(meteorgOutputInf.start\_gettingMeteorg())

**3.4 UML-діаграма класів.**

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.Діаграма класів (classdiagram) 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.

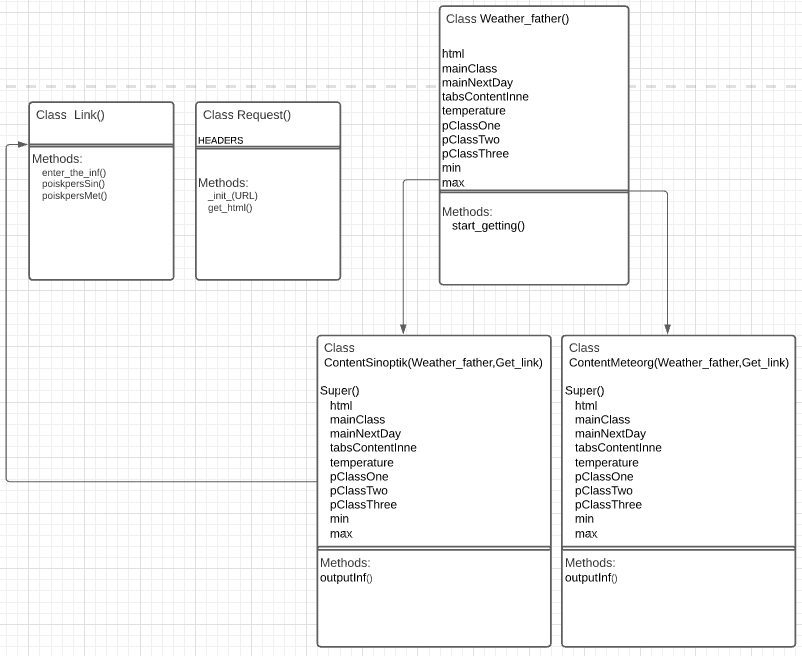


Рис. 3.7 UML діаграма класів.

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

Всі без винятку програмні продукти обробляють і виводять дані. Тільки в деяких інформація закладена спочатку, а в інших вона надходить ззовні. Черпати дані можна з файлів або безпосередньо з вводиться користувачем тексту. У другому випадку передбачається інтерактивний режим роботи. Виходить, користувач вводить інформацію, програма її обробляє і виводить на екран. У такому випадку говорять, що код не замкнутий на собі, а може «спілкуватися» з зовнішнім середовищем. Так, для введення інформації в останніх версіях Python input використовується.

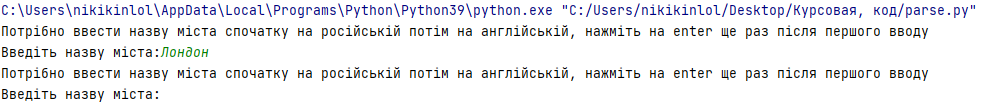
В програмі використовується можливість збереження введених даних у файл за допомогою функцій для роботи з файлами. Введена користувачем інформація (а саме назва локації див. пункт 2.2) обробляється, і в подальшому виводиться у консоль. Інформація про температуру повітря зберігається у файлах та зрівнюється, після чого виводиться оцінка точності прогнозу погоди для даного ресурсу

Усе що потрібно від користувача це правильний ввід назви населеного пункту.

Необхідно дотримуватись наступних умов:

* Ввід назви населеного пункту спочатку на російській а потім на англійській після чого натиснути на Enter
* Після вводу локації на російській необхідно натиснути на Enter ще раз, після цього виведеться уся доступна інформація про погоду на сайті sinoptik.ua
* Щоби також отримати інформацію з meteoprog.ua необхідно знову ввести назву локації але на цей раз на англійській

Приклад:



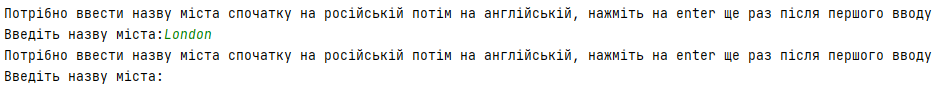
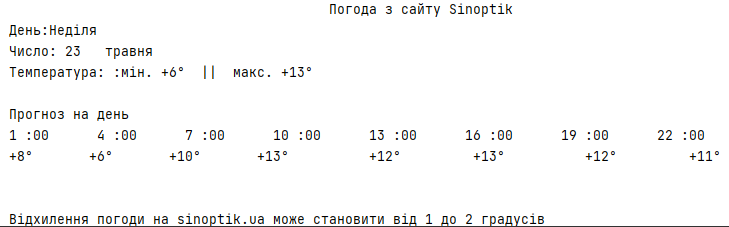


Рис. 5.1 Приклад вводу даних у консоль

Вивід:



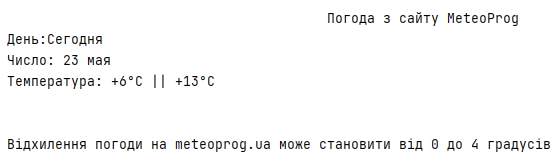


Рис. 5.2 Вивід даних з консолі

1. **ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

Тестування програми “ Агрегатор метеорологічних даних ” на основі функціонального тестування.

**Резюме (Summary)1:** Перевірка виводу інформації з інтернет ресурсів sinoptik.ua та meteoprog.ua

**Середовище виконання**: Windows 10.

**Опис (Description) –** Перевірка на вивід

Етапи для відтворення проблеми:

1. Відкриваю програму.

2. Запускаю код програми.

3. Ввожу назву населеного пункту спочатку на російській, нажимаю на Enter два рази, ввожу назву населеного пункту на англійській, нажимаю на Enter два рази

**Очікуваний результат виконання(Expected result)**: Консоль повинна вивести інформацію про погоду із джерел sinoptik.ua та meteoprog.ua

**Поточний результат виконання(Observed result)**: У консоль була виведена інформація про погоду із сайтів sinoptik.ua та meteoprog.ua

**Складність(Severity):** рівень середній

Тест пройшов успішно.

**Резюме (Summary)2:** Достовірність програми

**Середовище виконання**: Windows 10.

**Опис (Description) –** Перевірка програми на те чи інформація яку вона виводить аналогічна інформації на інтернет ресурсах sinoptik.ua та meteoprog.ua

Етапи для відтворення проблеми:

1. Відкриваю програму.

2. Запускаю код програми.

3. Ввожу назву населеного пункту спочатку на російській, нажимаю на Enter два рази, ввожу назву населеного пункту на англійській, нажимаю на Enter два рази

4. Отримую інформацію про погоду від програми

5. Захожу на сайт sinoptik.ua

6. Захожу на сайт meteoprog.ua

7. Зрівнюю отримані дані з тими що на сайті

**Очікуваний результат виконання(Expected result)**: Повний збіг погоди яку ми отримали від програми з тою що розташована на інтернет ресурсах

**Поточний результат виконання(Observed result)**: Програма видає погоду яка повністю ідентична погоді на сайтах sinoptik.ua та meteoprog.ua

**Складність(Severity):** рівень середній

Тест пройшов успішно.

1. **РОЗРОБКА ДОКУМЕНТАЦІЇ ДЛЯ СУПРОВОДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ**

Програма була створена за допомогою мови програмування python та таких бібліотек як: requests, BeautifulSoup, urllib3 (див. рис. 3.1). Для початку роботи необхідно завантажити необхідні файли з репозиторію:

<https://github.com/nikikinlol/dungeonMaster.git>

Файли які треба завантажити:

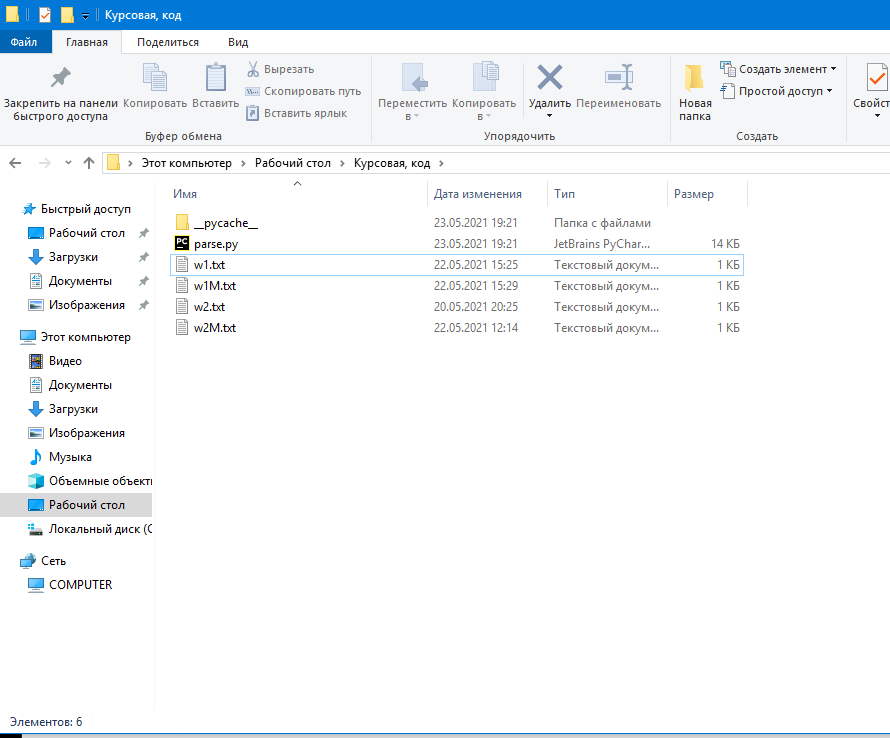


Рис 6.1 Файлова структура проекту

Вимоги:

* Операційна система: Windows 7/10
* Python: Version 3.9
* Pip 21.0
* Встановлені вищезгадані бібліотеки python

Для встановлення необхідних бібліотек запускаємо командну строку Windows:

Комбінація клавіш Ctrl+R і вводимо команду “cmd”

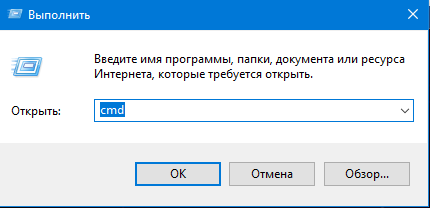


Рис 6.2 Ctrl+R

Далі запускається командна строка windows. Необхідно впевнитись що встановлений Python та Pip

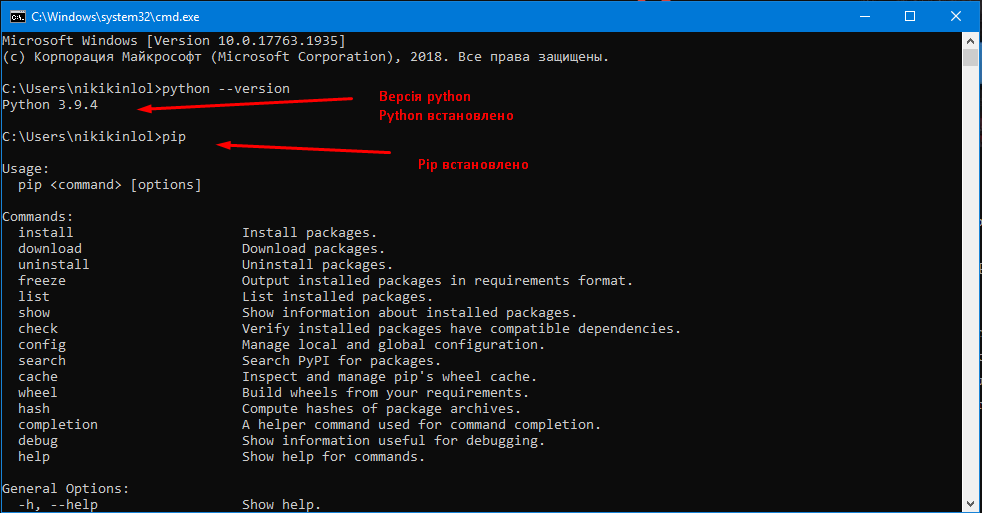


Рис 6.3 Перевірка на наявність python-pip

Встановлюємо потрібні бібліотеки:

* Для встановлення requests необхідно ввести команду pip install requests
* Для встановлення BeautifulSoup необхідно ввести команду pip install bs4
* Для встановлення urllib3 необхідно ввести команду pip install urllib3

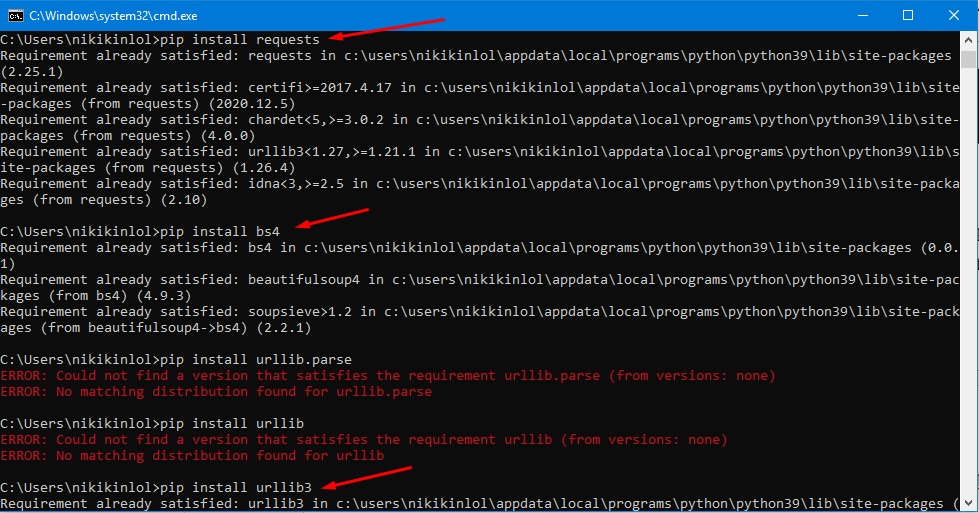


Рис 6.4 Встановлення бібліотек через командну строку

Для роботи з програмою та безпосередньо кодом цієї програми необхідно встановити Редактор коду. Для даної роботи був використаний редактор коду PyCharm.

Щоби скачати PyCharm потрібно перейти за цим посиланням:

<https://www.jetbrains.com/pycharm/>

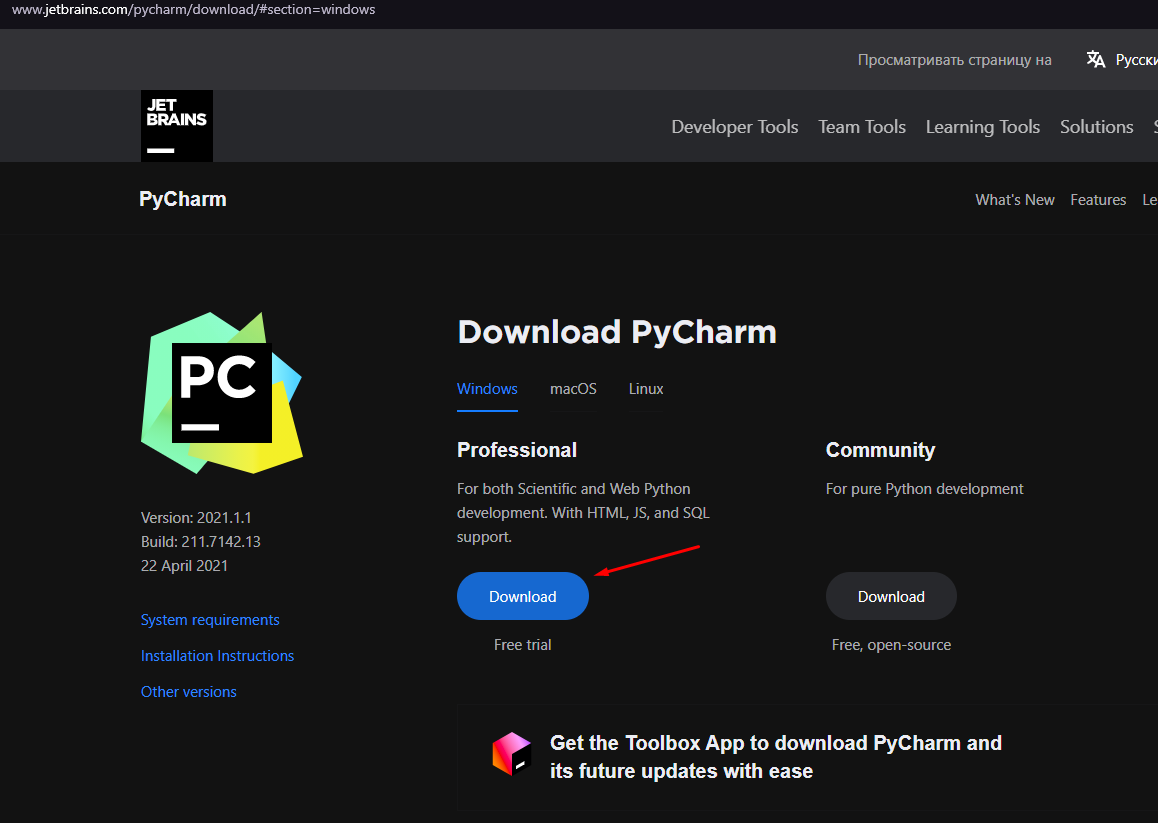


Рис 6.5 Сайт PyCharm

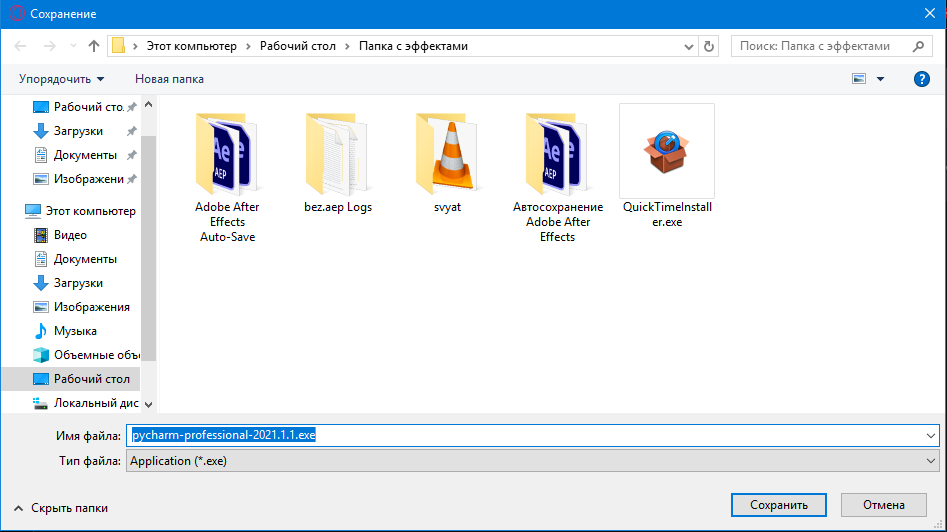


Рис 6.6 PyCharm.exe

При встановленні, бажано поставити галочку біля наступних пунктів:

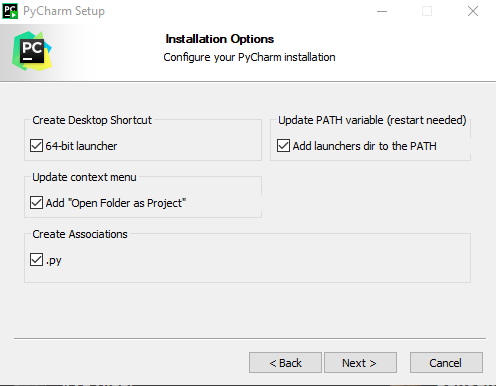


Рис 6.7 Встановлення редактору коду

Запускаємо PyCharm та імпортуємо встановлені раніше біліотеки

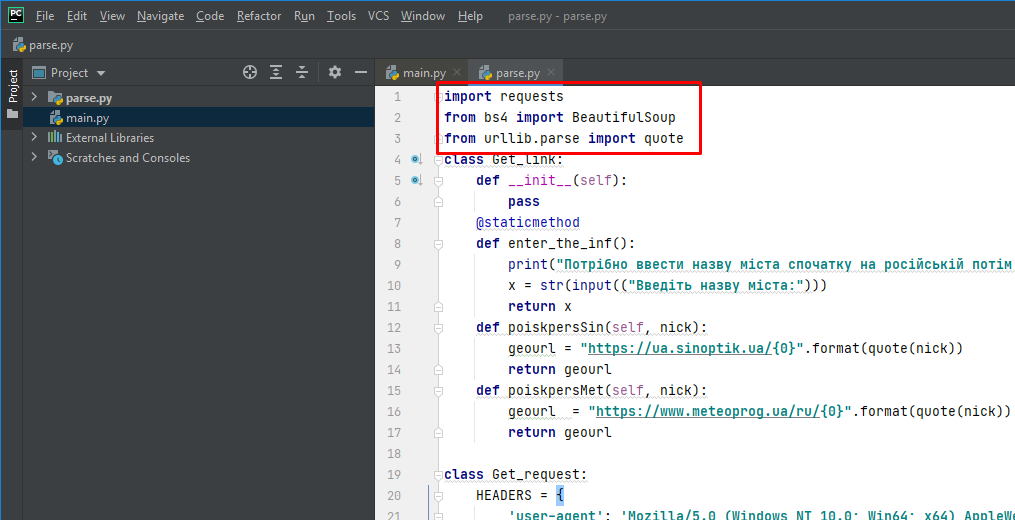


Рис 6.8 Імпорт бібліотек Python

**ВИСНОВОК**

В ході виконання курсової роботи на мові програмування Python я отримав наступний досвід:

* Працювати з редактором коду PyCharm
* Працювати з класами, об’єктами, методами класів
* Усвідомив що таке наслідування, поліморфізм та інкапсуляція
* Дізнався що таке парсинг
* Попрацював з бібліотеками request, BeautifulSoup та urllib3
* Навчився працювати з інформацією на сайті та як її отримувати
* Зрозумів що таке User-agent

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).

При роботі з класами зрозумів що вони дійсно полегшують життя, при написанні великого коду можна не боятися що глобальна змінна буде перезатерта у майбудньому шляхом присвоєння їй нового значення. Класи дуже корисні коли код доводиться підтримувати на протязі довгого часу. Нерідко буває що код був написаний 20 років тому назад але його підтримка спрощується у рази якщо він був написаний за стандартом ООП

**БІБЛІОГРАФІЧНИЙ СПИСОК**

1. Златопольский Д.М. Основи програмування на мові Python. - М .: ДМК Пресс, 2017. - 284 с.
2. Лутц М. Програмування на Python, том I, 4-е видання. - Пер. с англ. - СПб .: Символ-Плюс, 2011. - 992 с.
3. Гэддис Т. Починаємо програмувати на Python. - 4-е видання .: Пер. с англ. - СПб .: БХВ-Петербург, 2019. - 768 с.
4. Рейтц К., Шлюссер Т. Автостопом по Python. - СПб .: Питер, 2017. - 336 с .: іл. - (Серія «Бестселлеры O’Reilly»).
5. Гради Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на С++. — Бином, 1998.
6. Петер Коуд (Дэвид Норт, Марк Мэйфилд). Объектные модели. Стратегии, шаблоны и приложения.
7. Васильев О. Програмування мовою Python / О. Васильев//Тернопіль: Навчальна книга - Богдан, 2019. – С. 504.
8. Мэтиз, Э. Изучаем PYTHON.Программирование игр, визуализация данных, веб-приложения / Э. Мэтиз. - СПб.: Питер, 2017. - 496 c.
9. [www.metanit.com](http://www.metanit.com)
10. [www.learnpython.org](http://www.learnpython.org)
11. <https://docs.python.org/3/>
12. <https://www.python.org/doc/>
13. <https://realpython.com/documenting-python-code/>
14. <https://realpython.com/python-interface/>

Електронне посилання на репозиторій: https://github.com/nikikinlol/dungeonMaster.git

## ДОДАТОК

import requests

from bs4 import BeautifulSoup

from urllib.parse import quote

class Link:

def \_\_init\_\_(self):

pass

def enter\_the\_inf(self):

print("Потрібно ввести назву міста спочатку на російській потім на англійській, нажміть на enter ще раз після першого вводу")

x = str(input(("Введіть назву міста:")))

return x

def poiskpersSin(self, nick):

geourl = "https://ua.sinoptik.ua/{0}".format(quote(nick))

return geourl

def poiskpersMet(self, nick):

geourl = "https://www.meteoprog.ua/ru/{0}".format(quote(nick))

return geourl

class Request:

HEADERS = {

'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (HTML, like Gecko) '

'Chrome/88.0.4324.190 Safari/537.36',

'accept': '\*/\*'}

def \_\_init\_\_(self, URL1):

self.URL1 = URL1

def get\_html(self, params=None):

r = requests.get(self.URL1, headers=self.HEADERS, params=params)

return r.text