Міністерство освіти і науки України
Міжнародний гуманітарний університет
Кафедра комп’ютерної інженерії та інноваційних технологій

В.В. Педяш

ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

Методичні вказівки до практичних робіт
для здобувачів вищої освіти,
які навчаються за спеціальностями
галузі «Інформаційні технології»

Одеса 2025

Затверджено Вченою Радою Міжнародного гуманітарного університету.
Протокол № 5 від 20 січня 2025 р.

Педяш В.В. Об’єктно-орієнтоване програмування. Методичні вказівки до практичних робіт для здобувачів вищої освіти, які навчаються за спеціальностями галузі «Інформаційні технології» [Електронне видання]. Кафедра комп’ютерної інженерії та інноваційних технологій Міжнародного гуманітарного університету. Одеса, 2025. 55 с.

Дисципліна «Об’єктно-орієнтоване програмування» спрямована на формування системного розуміння принципів та методів розробки програмного забезпечення на основі об’єктно-орієнтованої парадигми програмування. Курс охоплює основні концепції об’єктно-орієнтованого підходу, зокрема поняття класу та об’єкта, інкапсуляцію, успадкування, поліморфізм та абстракцію. Значна увага приділяється моделюванню предметної області за допомогою класів і об’єктів, побудові ієрархій класів, організації взаємодії між програмними компонентами та повторному використанню програмного коду. У межах дисципліни розглядаються підходи до структуризації програмних систем, організації модулів і пакетів, а також принципи компонентної побудови програмного забезпечення.

ЗМІСТ

ТЕМА 1. ОСНОВИ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

Практична робота 1. Аналіз основних парадигм програмування та визначення особливостей об'єктно-орієнтованого підходу до розробки програмного забезпечення

Мета роботи – ознайомитися з основними парадигмами програмування, проаналізувати їх особливості та визначити характерні риси об'єктно-орієнтованого підходу до розробки програмного забезпечення.

Ключові положення

Програмування як галузь інформатики передбачає використання різних підходів до побудови програмного забезпечення, які називаються парадигмами програмування. Кожна парадигма визначає спосіб організації коду, структуру програм та принципи взаємодії між їх компонентами. Найбільш поширеними є процедурна, функціональна та об'єктно-орієнтована парадигми.

Процедурна парадигма базується на використанні послідовностей команд, процедур та функцій. Програма у цьому випадку розглядається як набір інструкцій, що виконуються у визначеному порядку. Основна увага приділяється алгоритмам і структурі керування виконанням програми. Дані і функції, що їх обробляють, зазвичай розглядаються окремо.

Функціональна парадигма орієнтована на використання функцій як основних елементів побудови програм. У такому підході функції розглядаються як математичні відображення, які не мають побічних ефектів. Основною ідеєю є обчислення результату через композицію функцій, що сприяє підвищенню надійності та передбачуваності програм.

Об'єктно-орієнтоване програмування базується на представленні програми у вигляді сукупності об'єктів, які взаємодіють між собою. Кожен об'єкт містить дані та методи для їх обробки. Основними поняттями є клас, об'єкт, інкапсуляція, успадкування та поліморфізм. Клас визначає структуру і поведінку об'єктів, а об'єкт є його конкретним екземпляром.

Інкапсуляція забезпечує об'єднання даних і методів у межах одного об'єкта та обмеження доступу до внутрішнього стану. Це дозволяє підвищити захищеність даних і спростити супровід програм. Успадкування дає змогу створювати нові класи на основі вже існуючих, розширюючи їх функціональність. Поліморфізм забезпечує можливість використання єдиного інтерфейсу для об'єктів різних типів.

Об'єктно-орієнтований підхід дозволяє реалізувати модульну структуру програмного забезпечення, забезпечує повторне використання коду та спрощує процес розробки складних систем. Він широко застосовується у сучасних мовах програмування, таких як Python, Java, C++.

Таким чином, вибір парадигми програмування визначає підхід до розробки програмного забезпечення. Об'єктно-орієнтоване програмування є одним із найбільш ефективних підходів для створення складних програмних систем завдяки своїй гнучкості та масштабованості.

Практичне завдання

1. Ознайомитися з основними парадигмами програмування.
2. Проаналізувати особливості процедурної парадигми програмування.
3. Проаналізувати особливості функціональної парадигми програмування.
4. Проаналізувати особливості об’єктно-орієнтованої парадигми програмування.
5. Визначити основні поняття об’єктно-орієнтованого програмування.
6. Порівняти підходи до організації програм у різних парадигмах.
7. Навести приклади задач, для яких доцільно використовувати кожну з парадигм.
8. Реалізувати простий приклад програми у процедурному стилі.
9. Реалізувати аналогічний приклад із використанням об’єктно-орієнтованого підходу.
10. Проаналізувати отримані результати та зробити висновки.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з теоретичними основами парадигм програмування. Доцільно звернути увагу на принципи організації коду у кожній парадигмі та їх вплив на структуру програмного забезпечення.

На першому етапі слід проаналізувати процедурну парадигму. Необхідно розглянути, як організовується програма у вигляді послідовності команд і функцій, а також визначити її переваги та обмеження. Доцільно звернути увагу на розділення даних і функцій.

Далі необхідно розглянути функціональну парадигму. Слід проаналізувати роль функцій, відсутність зміни стану та використання композиції функцій. Необхідно визначити, у яких випадках такий підхід є ефективним.

На наступному етапі потрібно детально дослідити об’єктно-орієнтований підхід. Необхідно розглянути поняття класу та об’єкта, а також принципи інкапсуляції, успадкування та поліморфізму. Доцільно звернути увагу на те, як ці принципи впливають на структуру програм.

Після теоретичного аналізу необхідно перейти до практичної частини. Спочатку слід реалізувати просту задачу у процедурному стилі, наприклад обробку даних або виконання обчислень. Далі потрібно реалізувати аналогічну задачу з використанням класів і об’єктів.

У процесі виконання роботи необхідно звернути увагу на відмінності у структурі програм, способах організації даних та повторному використанні коду. Доцільно проаналізувати, як об’єктно-орієнтований підхід дозволяє спростити розробку та супровід програм.

Після виконання завдання необхідно зробити висновки щодо ефективності різних парадигм програмування та визначити переваги об’єктно-орієнтованого підходу.

Контрольні питання

1. Що таке парадигма програмування?
2. Які основні парадигми програмування існують?
3. У чому полягає сутність процедурного підходу?
4. Які особливості функціонального програмування?
5. Що таке об’єктно-орієнтоване програмування?
6. Що таке клас і об’єкт?

7. У чому полягає принцип інкапсуляції?
8. Що таке успадкування?
9. У чому сутність поліморфізму?
10. Які переваги об’єктно-орієнтованого підходу?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– короткий опис парадигм програмування;
– результати аналізу особливостей кожної парадигми;
– приклади реалізації програм у різних парадигмах;
– порівняльний аналіз підходів;
– висновки.

Практична робота 2. Розробка простої моделі предметної області у вигляді системи класів та об’єктів

Мета роботи – набути практичних навичок побудови простої моделі предметної області засобами об’єктно-орієнтованого програмування шляхом визначення класів, їх атрибутів, методів та створення об’єктів.

Ключові положення

Одним із важливих етапів розробки програмного забезпечення є побудова моделі предметної області. Під предметною областю розуміють сукупність об’єктів, процесів, зв’язків і правил, які характерні для певної сфери діяльності. У програмуванні така область подається у вигляді абстрактної моделі, яка відображає її основні сутності та взаємодії між ними.

У межах об’єктно-орієнтованого підходу модель предметної області будується за допомогою класів та об’єктів. Клас використовується для опису сутності, її властивостей і доступних дій. Об’єкт є конкретним представником класу, який має певні значення атрибутів і може виконувати визначені методи. Такий підхід дає змогу подати структуру реального світу у вигляді програмних конструкцій.

Проєктування моделі предметної області передбачає визначення основних сутностей. Для кожної сутності необхідно встановити, які характеристики є істотними, а які дії можуть виконуватися над її даними. Характеристики подаються у вигляді атрибутів класу, а дії – у вигляді методів. Важливо, щоб модель не містила надлишкових елементів і водночас забезпечувала достатню повноту опису предметної області.

Під час побудови системи класів слід враховувати зв’язки між сутностями. У простих моделях це можуть бути зв’язки типу один до одного, один до багатьох або включення одного об’єкта до складу іншого. Такі зв’язки дають змогу формувати цілісну структуру програмної моделі та відображати відношення між елементами предметної області.

Модель предметної області повинна бути логічною, зрозумілою та придатною до подальшого розширення. Тому на початковому етапі доцільно виділяти лише основні

сутності та найважливіші зв’язки між ними. Надмірне ускладнення моделі ускладнює її реалізацію та знижує наочність.

У процесі реалізації моделі важливу роль відіграє правильне іменування класів, атрибутів і методів. Назви повинні відображати зміст відповідних елементів і бути зрозумілими з погляду предметної області. Це спрощує читання програми, її супровід і подальше розширення.

Практична реалізація моделі предметної області дає змогу перейти від абстрактного опису до програмного представлення. Для цього створюються класи, визначаються їх конструктори, атрибути та методи, після чого формуються об’єкти і перевіряється коректність їх взаємодії. Таке моделювання є основою для побудови більш складних програмних систем.

Таким чином, розробка простої моделі предметної області у вигляді системи класів та об’єктів є важливим етапом засвоєння об’єктно-орієнтованого програмування. Вона дає змогу навчитися формалізувати реальні сутності та подавати їх у вигляді програмних конструкцій.

Практичне завдання

1. Обрати просту предметну область для моделювання.
2. Визначити основні сутності предметної області.
3. Виділити характеристики кожної сутності.
4. Визначити дії, які можуть виконуватися для кожної сутності.
5. Розробити структуру системи класів.
6. Створити класи для основних сутностей предметної області.
7. Реалізувати атрибути і методи для кожного класу.
8. Створити об’єкти розроблених класів.
9. Організувати взаємодію між об’єктами.
10. Перевірити працездатність моделі на простому прикладі.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно обрати просту предметну область, для якої можна побудувати модель засобами об’єктно-орієнтованого програмування. Доцільно використовувати зрозумілі приклади, наприклад бібліотеку, навчальну групу, магазин, транспортну систему або систему обліку товарів.

На першому етапі слід визначити основні сутності предметної області. Необхідно виділити об’єкти, які мають самостійне значення у межах вибраної системи. Наприклад, для моделі бібліотеки такими сутностями можуть бути книга, читач і бібліотека.

Далі потрібно встановити, які характеристики мають вибрані сутності. Для цього необхідно визначити атрибути класів, які будуть описувати стан об’єктів. Наприклад, для класу книга це можуть бути назва, автор, рік видання, а для класу читач – прізвище, ім’я, номер читацького квитка.

Після цього необхідно визначити дії, які повинні виконувати об’єкти. Такі дії реалізуються у вигляді методів класу. Наприклад, для книги це може бути виведення інформації, а для бібліотеки – додавання книги або реєстрація читача. Методи повинні відповідати логіці предметної області.

На наступному етапі слід спроєктувати систему класів. Необхідно визначити, які класи є основними, а які можуть використовуватися для організації зв’язків між об’єктами. Якщо один об’єкт містить інші об’єкти, це потрібно врахувати при побудові моделі.

Після проєктування необхідно реалізувати класи засобами мови програмування. Для кожного класу доцільно створити конструктор, який забезпечує початкове задання значень атрибутів. Також потрібно реалізувати методи, які відображають поведінку об’єктів.

Далі необхідно створити кілька об’єктів кожного класу та перевірити їх взаємодію. У процесі виконання роботи слід переконатися, що атрибути набувають правильних значень, методи виконуються коректно, а зв’язки між об’єктами відповідають структурі предметної області.

Після завершення реалізації необхідно проаналізувати отриману модель. Доцільно оцінити, наскільки повно вона відображає вибрану предметну область, чи не містить зайвих елементів і чи може бути розширена у подальшому.

Контрольні питання

1. Що таке предметна область?
2. Для чого виконується моделювання предметної області?
3. Яким чином предметна область подається в об’єктно-орієнтованому програмуванні?
4. Що таке клас у моделі предметної області?
5. Що таке об’єкт у моделі предметної області?
6. Як визначаються атрибути класу?
7. Як визначаються методи класу?
8. Які зв’язки можуть існувати між об’єктами?
9. Чому модель предметної області повинна бути простою та логічною?
10. Які переваги дає моделювання предметної області засобами класів та об’єктів?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– назва вибраної предметної області;
– опис основних сутностей предметної області;
– перелік розроблених класів, їх атрибутів і методів;
– приклади створених об’єктів;
– опис взаємодії між об’єктами;
– аналіз отриманих результатів;
– висновки.

Практична робота 3. Створення класів у Python та реалізація атрибутів і методів класу

Мета роботи – набути практичних навичок створення класів у мові програмування Python, визначення атрибутів і реалізації методів для опису стану та поведінки об’єктів.

Ключові положення

У мові програмування Python об’єктно-орієнтований підхід реалізується через використання класів і об’єктів. Клас є шаблоном, що визначає структуру даних та набір операцій, які можуть виконуватися над цими даними. Об’єкт є конкретним екземпляром класу, який має власні значення атрибутів.

Опис класу у Python здійснюється за допомогою ключового слова class. У межах класу визначаються атрибути та методи. Атрибути описують стан об’єкта, а методи – його поведінку. Для ініціалізації об’єкта використовується спеціальний метод __init__, який викликається під час створення нового екземпляра класу.

Атрибути можуть бути визначені як змінні екземпляра або як змінні класу. Змінні екземпляра зберігають індивідуальні дані кожного об’єкта, тоді як змінні класу є спільними для всіх об’єктів цього класу. Доступ до атрибутів здійснюється через посилання на об’єкт за допомогою синтаксису self.

Методи класу визначаються як функції, що описані всередині класу. Першим параметром методу зазвичай є self, який посилається на поточний об’єкт. Методи можуть змінювати стан об’єкта, обробляти його дані або повертати результати обчислень.

Важливим аспектом є правильна організація класу. Атрибути і методи повинні бути логічно пов’язані між собою і відображати сутність об’єкта. Доцільно забезпечувати мінімальну залежність між класами та уникати дублювання коду.

У Python також підтримується інкапсуляція, яка дозволяє обмежувати доступ до атрибутів. Для цього використовуються умовні позначення імен, зокрема один або два символи підкреслення на початку імені. Це дає змогу відокремити внутрішню реалізацію класу від його зовнішнього інтерфейсу.

Створення класів і реалізація їх методів є основою побудови об’єктно-орієнтованих програм. Правильне використання цих механізмів дозволяє створювати структурований, зрозумілий і масштабований програмний код.

Практичне завдання

1. Створити новий програмний файл мовою Python.
2. Описати клас, що відповідає певній предметній області.
3. Визначити атрибути класу.
4. Реалізувати метод __init__ для ініціалізації об’єкта.
5. Додати методи для обробки даних об’єкта.
6. Створити декілька об’єктів класу.
7. Присвоїти значення атрибутам об’єктів.
8. Викликати методи для кожного об’єкта.
9. Вивести результати роботи програми.
10. Перевірити коректність роботи реалізованого класу.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно обрати просту предметну область, яка буде реалізована у вигляді класу. Це може бути модель об’єкта, наприклад студент, товар, автомобіль або інша сутність, що має набір характеристик і дій.

На першому етапі слід створити файл програми і визначити клас за допомогою ключового слова class. Необхідно задати ім’я класу, яке повинно відображати його призначення.

Далі потрібно визначити атрибути класу. Для цього у методі __init__ слід описати параметри, які передаються під час створення об’єкта, та присвоїти їх змінним екземпляра через self. Це дозволяє кожному об’єкту мати власний набір даних.

Після цього необхідно реалізувати методи класу. Методи повинні виконувати дії, які відповідають логіці предметної області. Наприклад, це може бути обчислення значень, зміна стану об’єкта або виведення інформації.

На наступному етапі потрібно створити об’єкти класу. Для цього викликається конструктор класу з передаванням необхідних параметрів. Кожен об’єкт зберігає власні значення атрибутів.

Далі необхідно викликати методи для створених об’єктів і перевірити результати їх роботи. У процесі виконання роботи слід звернути увагу на правильність доступу до атрибутів і коректність реалізації методів.

У разі виникнення помилок необхідно перевірити правильність синтаксису, відповідність імен атрибутів і параметрів, а також логіку роботи методів. Доцільно використовувати виведення проміжних результатів для перевірки роботи програми.

Після завершення роботи необхідно проаналізувати отримані результати та оцінити, наскільки реалізований клас відповідає поставленому завданню і чи може бути розширений у подальшому.

Контрольні питання

1. Що таке клас у Python?
2. Що таке об’єкт?
3. Яке призначення методу __init__?
4. Що таке атрибути класу?
5. Чим відрізняються змінні класу і змінні екземпляра?
6. Що таке метод класу?
7. Яка роль параметра self?
8. Як створюється об’єкт класу?
9. Як викликаються методи об’єкта?
10. Які помилки можуть виникати при створенні класів у Python?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис розробленого класу;

– перелік атрибутів і методів класу;
– приклади створених об’єктів;
– результати виконання програми;
– аналіз отриманих результатів;
– висновки.

Практична робота 4. Створення об’єктів класу та дослідження взаємодії об’єктів у програмі

Мета роботи – набути практичних навичок створення об’єктів класів, організації їх взаємодії та дослідження обміну даними між об’єктами у програмі.

Ключові положення

У об’єктно-орієнтованому програмуванні об’єкти є основними елементами програмної системи, які взаємодіють між собою для реалізації необхідної функціональності. Об’єкт створюється на основі класу та містить власні значення атрибутів, що визначають його стан.

Взаємодія між об’єктами здійснюється через виклик методів і передачу даних. Один об’єкт може викликати метод іншого об’єкта, передаючи йому необхідні параметри. Такий підхід дозволяє організувати обмін інформацією та координувати виконання дій у програмі.

Одним із способів організації взаємодії є передавання об’єктів як параметрів методів. Це дозволяє одному об’єкту використовувати дані або функціональність іншого. У результаті формується зв’язана структура об’єктів, яка відображає логіку предметної області.

Важливим аспектом є керування станом об’єктів. Зміна стану одного об’єкта може впливати на поведінку іншого. Тому необхідно забезпечити коректну послідовність викликів методів та узгодженість даних між об’єктами.

У процесі взаємодії об’єктів доцільно дотримуватися принципу мінімальної залежності. Кожен об’єкт повинен виконувати лише ті функції, які відповідають його призначенню. Надмірна залежність між об’єктами ускладнює структуру програми та її супровід.

Для перевірки взаємодії об’єктів використовуються тестові сценарії, які дозволяють відтворити типові ситуації використання програми. Це дає змогу оцінити коректність роботи системи та виявити можливі помилки у взаємодії компонентів.

Таким чином, створення об’єктів та організація їх взаємодії є ключовими етапами розробки об’єктно-орієнтованих програм. Від правильності їх реалізації залежить цілісність і ефективність програмної системи.

Практичне завдання

1. Використати раніше створений клас або розробити новий.
2. Створити декілька об’єктів класу.
3. Ініціалізувати атрибути кожного об’єкта.
4. Реалізувати методи для взаємодії між об’єктами.
5. Організувати передачу об’єктів як параметрів методів.

6. Реалізувати обмін даними між об’єктами.
7. Змоделювати простий сценарій взаємодії об’єктів.
8. Вивести результати взаємодії у зручному вигляді.
9. Проаналізувати зміну стану об’єктів у процесі роботи програми.
10. Перевірити коректність реалізації взаємодії.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити клас або систему класів, які будуть використані для моделювання взаємодії. Доцільно використовувати приклади, у яких об’єкти природно взаємодіють між собою, наприклад покупець і товар, студент і курс, користувач і система.

На першому етапі необхідно створити кілька об’єктів одного або різних класів. Для кожного об’єкта слід задати початкові значення атрибутів, що визначають його стан.

Далі потрібно реалізувати методи, які забезпечують взаємодію між об’єктами. Це можуть бути методи, що приймають інший об’єкт як параметр, або методи, що змінюють стан об’єкта на основі даних іншого об’єкта.

На наступному етапі слід організувати передачу об’єктів між методами. Необхідно переконатися, що доступ до атрибутів інших об’єктів здійснюється коректно, а зміни стану відбуваються відповідно до логіки програми.

У процесі виконання роботи доцільно реалізувати кілька сценаріїв взаємодії. Наприклад, один об’єкт може змінювати стан іншого, або кілька об’єктів можуть взаємодіяти у певній послідовності. Це дозволяє перевірити гнучкість і коректність реалізації.

Особливу увагу слід приділити узгодженості даних. Після виконання кожної операції необхідно перевіряти, чи відповідає стан об’єктів очікуваному результату. Для цього доцільно виводити проміжні значення атрибутів.

У разі виникнення помилок необхідно перевірити правильність викликів методів, передачі параметрів та логіку взаємодії об’єктів. Також слід переконатися, що кожен об’єкт виконує лише свої функції.

Після завершення роботи необхідно проаналізувати отримані результати та оцінити ефективність організації взаємодії об’єктів.

Контрольні питання

1. Що таке об’єкт у об’єктно-орієнтованому програмуванні?
2. Як створюється об’єкт класу?
3. Яким чином об’єкти можуть взаємодіяти між собою?
4. Що означає передавання об’єкта як параметра?
5. Як змінюється стан об’єкта?
6. Які існують способи організації взаємодії об’єктів?
7. Чому важливо забезпечувати мінімальну залежність між об’єктами?
8. Як перевірити коректність взаємодії об’єктів?
9. Які помилки можуть виникати при взаємодії об’єктів?
10. Які переваги дає використання об’єктів у програмі?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис використаних класів;
– приклади створених об’єктів;
– опис реалізованої взаємодії;
– результати виконання програми;
– аналіз зміни стану об’єктів;
– висновки.

ТЕМА 2. КЛАСИ, МЕТОДИ ТА КОНСТРУЮВАННЯ ОБ'ЄКТІВ

Практична робота 5. Реалізація методів класу та дослідження механізму передавання параметрів і повернення результатів

Мета роботи – набути практичних навичок реалізації методів класу, дослідити механізм передавання параметрів у методи та повернення результатів їх виконання.

Ключові положення

У об'єктно-орієнтованому програмуванні методи класу визначають поведінку об'єктів та забезпечують обробку їх даних. Метод є функцією, що входить до складу класу і виконується для конкретного об'єкта. Через методи реалізується доступ до атрибутів об'єкта та їх зміна.

Методи можуть приймати параметри, які передаються під час їх виклику. Передавання параметрів дозволяє передавати в метод необхідні дані для виконання обчислень або зміни стану об'єкта. У Python параметри передаються за посиланням на об'єкт, що означає, що метод працює з фактичними значеннями, а не їх копіями у традиційному розумінні.

Результатом виконання методу може бути повернення значення. Для цього використовується оператор return. Повернене значення може бути використане у подальших обчисленнях або для виведення результату роботи програми. Якщо оператор return відсутній, метод повертає спеціальне значення None.

Методи можуть виконувати різні функції: змінювати стан об'єкта, обчислювати значення на основі атрибутів, взаємодіяти з іншими об'єктами або повертати інформацію про стан об'єкта. Важливо, щоб методи відповідали логіці предметної області та не виконували надлишкових дій.

Під час реалізації методів необхідно враховувати типи параметрів, їх кількість і порядок передавання. Некоректне передавання параметрів може призвести до помилок виконання програми. Тому важливо забезпечити узгодженість між описом методу та його використанням.

Дослідження механізму передавання параметрів передбачає аналіз того, як зміна значень усередині методу впливає на об'єкти поза ним. Це дозволяє зрозуміти особливості роботи з об'єктами і змінними у Python.

Таким чином, реалізація методів класу та дослідження механізму передавання параметрів і повернення результатів є важливими для розуміння поведінки об'єктів і побудови коректних програм.

Практичне завдання

1. Використати раніше створений клас або розробити новий.
2. Реалізувати методи, що змінюють стан об'єкта.
3. Реалізувати методи, що приймають параметри.
4. Реалізувати методи, що повертають результати.
5. Викликати методи з різними параметрами.
6. Дослідити вплив параметрів на результат виконання методів.

7. Реалізувати приклади використання оператора return.

8. Порівняти методи, що змінюють стан об’єкта, і методи, що лише повертають значення.

9. Вивести результати роботи методів.

10. Проаналізувати отримані результати.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити клас, для якого будуть реалізовані методи. Доцільно використовувати приклади, у яких об’єкт має як стан, так і поведінку, наприклад рахунок, товар або студент.

На першому етапі слід реалізувати методи, що змінюють стан об’єкта. Це можуть бути методи, які змінюють значення атрибутів на основі переданих параметрів або внутрішньої логіки.

Далі необхідно реалізувати методи, що приймають параметри. Слід звернути увагу на правильність оголошення параметрів та їх використання у тілі методу. Доцільно перевірити роботу методу при різних значеннях параметрів.

На наступному етапі потрібно реалізувати методи, що повертають значення. Для цього використовується оператор return. Необхідно переконатися, що повернене значення відповідає очікуваному результату.

У процесі виконання роботи слід дослідити, як передавання параметрів впливає на об’єкти. Зокрема, необхідно перевірити, чи змінюється стан об’єкта при передаванні його у метод, а також як поводяться прості та складні типи даних.

Далі потрібно виконати виклики методів з різними наборами параметрів і проаналізувати отримані результати. Доцільно порівняти методи, які змінюють стан об’єкта, з методами, що лише повертають значення без зміни стану.

Особливу увагу слід приділити правильності використання оператора return. Необхідно переконатися, що у всіх випадках методи повертають коректні значення або None, якщо повернення не передбачене.

У разі виникнення помилок необхідно перевірити відповідність кількості переданих параметрів, правильність їх використання та логіку роботи методів.

Після завершення роботи необхідно проаналізувати отримані результати та зробити висновки щодо механізму передавання параметрів і повернення результатів.

Контрольні питання

1. Що таке метод класу?
2. Яка роль параметрів у методах?
3. Як передаються параметри у Python?
4. Що таке оператор return?
5. Яке значення повертається, якщо return відсутній?
6. Які методи змінюють стан об’єкта?
7. Які методи лише повертають значення?
8. Як впливає передавання об’єкта у метод на його стан?
9. Які помилки можуть виникати при передаванні параметрів?
10. Як перевірити коректність роботи методів?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис реалізованих методів;
– приклади виклику методів;
– результати виконання програми;
– аналіз передавання параметрів і повернення результатів;
– висновки.

Практична робота 6. Реалізація різних типів методів у Python: методів екземпляра, класових і статичних методів

Мета роботи – набути практичних навичок реалізації різних типів методів у Python, дослідити особливості їх використання та відмінності між методами екземпляра, класовими і статичними методами.

Ключові положення

У мові програмування Python методи класу можуть бути різних типів залежно від способу їх виклику та доступу до даних. Основними типами є методи екземпляра, класові та статичні методи. Кожен із цих типів має своє призначення і використовується для реалізації різних аспектів поведінки класу.

Методи екземпляра є найбільш поширеними. Вони працюють із конкретним об’єктом і мають доступ до його атрибутів. Першим параметром такого методу є self, який посилається на поточний об’єкт. За допомогою методів екземпляра можна змінювати стан об’єкта або отримувати інформацію про нього.

Класові методи призначені для роботи з класом як цілим. Вони визначаються з використанням декоратора @classmethod і приймають параметр cls, який посилається на сам клас. Класові методи можуть змінювати стан класу або створювати нові об’єкти. Вони часто використовуються для реалізації альтернативних конструкторів.

Статичні методи не мають доступу ні до об’єкта, ні до класу. Вони визначаються за допомогою декоратора @staticmethod і не приймають параметри self або cls. Такі методи використовуються для виконання допоміжних операцій, які логічно пов’язані з класом, але не потребують доступу до його стану.

Вибір типу методу залежить від того, з якими даними він працює. Якщо метод використовує або змінює атрибути об’єкта, слід використовувати метод екземпляра. Якщо необхідно працювати з даними класу або створювати об’єкти, доцільно застосовувати класові методи. Якщо метод виконує незалежні обчислення, використовуються статичні методи.

Правильне використання різних типів методів дозволяє зробити структуру класу більш зрозумілою та логічною. Це сприяє кращій організації коду і підвищує його повторне використання.

Таким чином, методи екземпляра, класові та статичні методи є важливими інструментами реалізації поведінки класів у Python. Їх використання залежить від задачі та характеру обробки даних.

Практичне завдання

1. Створити клас у мові Python.
2. Реалізувати метод екземпляра.
3. Реалізувати класовий метод із використанням декоратора @classmethod.
4. Реалізувати статичний метод із використанням декоратора @staticmethod.
5. Викликати метод екземпляра через об’єкт.
6. Викликати класовий метод через клас і через об’єкт.
7. Викликати статичний метод через клас і через об’єкт.
8. Дослідити доступ кожного типу методу до даних.
9. Порівняти результати роботи різних типів методів.
10. Зробити висновки щодо доцільності використання кожного типу методів.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно створити клас, який міститиме різні типи методів. Доцільно обрати приклад, у якому є як дані об’єкта, так і спільні дані для всіх об’єктів класу.

На першому етапі слід реалізувати метод екземпляра. Він повинен використовувати параметр self і працювати з атрибутами об’єкта. Необхідно перевірити, що метод може змінювати стан об’єкта або повертати його значення.

Далі необхідно реалізувати класовий метод. Для цього використовується декоратор @classmethod. Метод повинен приймати параметр cls і працювати з атрибутами класу. Доцільно реалізувати метод, який створює новий об’єкт або змінює спільні дані класу.

На наступному етапі потрібно реалізувати статичний метод. Для цього використовується декоратор @staticmethod. Такий метод не повинен використовувати self або cls. Він виконує допоміжні обчислення, пов’язані з класом.

Після реалізації методів необхідно виконати їх виклик різними способами. Метод екземпляра викликається через об’єкт, класовий і статичний методи можуть викликатися як через клас, так і через об’єкт.

У процесі виконання роботи слід дослідити, до яких даних має доступ кожен тип методу. Необхідно перевірити, що метод екземпляра працює з атрибутами об’єкта, класовий метод – з атрибутами класу, а статичний метод не має доступу до стану об’єкта чи класу.

Далі потрібно порівняти результати роботи методів і визначити їх відмінності. Особливу увагу слід приділити ролі параметрів self і cls.

У разі виникнення помилок необхідно перевірити правильність використання декораторів, параметрів методів і способів їх виклику.

Після завершення роботи необхідно зробити висновки щодо особливостей реалізації різних типів методів у Python.

Контрольні питання

1. Які типи методів існують у Python?
2. Що таке метод екземпляра?
3. Яка роль параметра self?
4. Що таке класовий метод?
5. Яка роль параметра cls?
6. Що таке статичний метод?
7. Чим відрізняється класовий метод від статичного?
8. Як викликаються різні типи методів?
9. До яких даних має доступ кожен тип методу?
10. У яких випадках доцільно використовувати кожен тип методів?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис реалізованого класу;
– приклади методів різних типів;
– результати виконання програми;
– порівняльний аналіз методів;
– висновки.

Практична робота 7. Створення конструкторів класу та дослідження механізму ініціалізації об’єктів

Мета роботи – набути практичних навичок створення конструкторів класу в Python, дослідити механізм ініціалізації об’єктів та особливості передавання параметрів під час їх створення.

Ключові положення

У об’єктно-орієнтованому програмуванні створення об’єкта супроводжується процесом його ініціалізації, під час якого встановлюються початкові значення атрибутів. У Python цей процес реалізується за допомогою спеціального методу __init__, який викликається автоматично під час створення нового об’єкта класу.

Метод __init__ є конструктором класу, який відповідає за початкове налаштування об’єкта. Він приймає параметри, що передаються під час створення об’єкта, і присвоює їх атрибутам через посилання self. Це дозволяє кожному об’єкту мати власний стан.

Конструктор може приймати довільну кількість параметрів або не приймати їх взагалі. У разі відсутності параметрів ініціалізація виконується з використанням значень за замовчуванням. Це дозволяє створювати об’єкти з різним рівнем деталізації.

У Python також можна використовувати значення параметрів за замовчуванням, що дає змогу спростити створення об’єктів. У такому випадку частина параметрів може не передаватися під час створення об’єкта, а їх значення встановлюються автоматично.

Важливим аспектом є правильна організація конструктора. Він повинен забезпечувати коректну ініціалізацію всіх необхідних атрибутів і не виконувати надлишкових дій. Конструктор не повинен містити складної логіки, яка не пов'язана безпосередньо з ініціалізацією об'єкта.

Під час створення об'єкта виконується виклик конструктора з передаванням відповідних параметрів. У разі некоректного передавання параметрів можуть виникати помилки, пов'язані з кількістю або типами аргументів. Тому важливо забезпечити відповідність між описом конструктора та його використанням.

Дослідження механізму ініціалізації передбачає аналіз того, як значення параметрів передаються в конструктор і як вони впливають на стан об'єкта. Це дозволяє зрозуміти процес створення об'єктів і забезпечити їх правильну роботу.

Таким чином, створення конструкторів класу та дослідження механізму ініціалізації є важливими етапами розробки об'єктно-орієнтованих програм у Python.

Практичне завдання

1. Створити клас у мові Python.
2. Реалізувати конструктор __init__ без параметрів.
3. Реалізувати конструктор із параметрами.
4. Використати значення параметрів за замовчуванням.
5. Створити об'єкти з різними наборами параметрів.
6. Ініціалізувати атрибути об'єктів.
7. Вивести значення атрибутів для кожного об'єкта.
8. Дослідити вплив параметрів на стан об'єкта.
9. Обробити можливі помилки при створенні об'єктів.
10. Проаналізувати результати виконання програми.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно створити клас, який міститиме конструктор для ініціалізації об'єктів. Доцільно обрати приклад, у якому об'єкт має кілька атрибутів, що визначають його стан.

На першому етапі слід реалізувати простий конструктор без параметрів. У такому випадку значення атрибутів можуть задаватися безпосередньо у тілі методу __init__.

Далі необхідно реалізувати конструктор із параметрами. Потрібно визначити параметри, які будуть передаватися під час створення об'єкта, і присвоїти їх відповідним атрибутам через self.

На наступному етапі слід використати значення параметрів за замовчуванням. Це дозволить створювати об'єкти без передавання всіх параметрів. Необхідно перевірити, як змінюється стан об'єкта залежно від переданих значень.

Після цього потрібно створити декілька об'єктів із різними наборами параметрів. Це дозволить дослідити гнучкість конструктора та його поведінку у різних ситуаціях.

У процесі виконання роботи необхідно вивести значення атрибутів кожного об'єкта та переконатися у правильності їх ініціалізації. Доцільно використовувати додаткові методи для виведення інформації.

Особливу увагу слід приділити перевірці правильності передавання параметрів. У разі виникнення помилок необхідно перевірити відповідність кількості та порядку аргументів.

Далі потрібно проаналізувати, як зміна параметрів впливає на стан об’єкта. Це дозволить краще зрозуміти механізм ініціалізації.

Після завершення роботи необхідно зробити висновки щодо особливостей реалізації конструкторів у Python.

Контрольні питання

1. Що таке конструктор класу?
2. Яке призначення методу __init__?
3. Коли викликається конструктор?
4. Чи може конструктор не мати параметрів?
5. Що таке параметри за замовчуванням?
6. Як передаються параметри у конструктор?
7. Як ініціалізуються атрибути об’єкта?
8. Які помилки можуть виникати при створенні об’єктів?
9. Як перевірити правильність ініціалізації об’єкта?
10. Які переваги використання конструктора?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис реалізованого конструктора;
– приклади створених об’єктів;
– результати ініціалізації атрибутів;
– аналіз роботи конструктора;
– висновки.

Практична робота 8. Реалізація класів із використанням конструкторів із параметрами

Мета роботи – набути практичних навичок реалізації класів із використанням конструкторів із параметрами, дослідити особливості передавання аргументів та їх вплив на ініціалізацію об’єктів.

Ключові положення

У об’єктно-орієнтованому програмуванні конструктор класу забезпечує початкову ініціалізацію об’єкта. У Python для цього використовується метод __init__, який викликається автоматично під час створення об’єкта та дозволяє передавати параметри для встановлення початкового стану.

Конструктор із параметрами дає змогу створювати об’єкти з різними значеннями атрибутів. Це забезпечує гнучкість при роботі з класами та дозволяє уникнути необхідності додаткового налаштування об’єкта після його створення.

Параметри конструктора визначаються у сигнатурі методу __init__. Кількість і типи параметрів залежать від моделі предметної області. Під час створення об’єкта аргументи передаються у конструктор і присвоюються атрибутам через self.

У Python підтримується передавання позиційних і іменованих аргументів. Іменовані аргументи дозволяють явно вказувати, які значення передаються відповідним параметрам, що підвищує читабельність коду і зменшує ймовірність помилок.

Важливим аспектом є перевірка коректності переданих параметрів. Невідповідність кількості або типів аргументів може призвести до помилок під час виконання програми. Тому необхідно забезпечити узгодженість між описом конструктора та його використанням.

Конструктор може також містити просту логіку для перевірки або обробки переданих параметрів. Наприклад, можна перевіряти допустимість значень або виконувати їх перетворення перед присвоєнням атрибутам.

Реалізація класів із використанням конструкторів із параметрами дозволяє створювати більш гнучкі та універсальні програмні компоненти, які легко адаптуються до різних умов використання.

Таким чином, використання параметризованих конструкторів є важливим елементом об’єктно-орієнтованого програмування у Python.

Практичне завдання

1. Створити клас у мові Python.
2. Реалізувати конструктор із параметрами.
3. Визначити атрибути, які ініціалізуються через параметри.
4. Використати позиційні аргументи при створенні об’єктів.
5. Використати іменовані аргументи при створенні об’єктів.
6. Створити декілька об’єктів із різними значеннями параметрів.
7. Реалізувати перевірку коректності переданих параметрів.
8. Вивести значення атрибутів об’єктів.
9. Проаналізувати вплив параметрів на стан об’єктів.
10. Перевірити роботу програми.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити клас, який буде реалізований із використанням конструктора з параметрами. Доцільно обрати приклад, у якому об’єкт має декілька характеристик, що задаються під час створення.

На першому етапі слід описати клас і реалізувати метод __init__ з параметрами. Необхідно визначити перелік параметрів, які відповідають атрибутам об’єкта, та присвоїти їх змінним екземпляра через self.

Далі потрібно створити об’єкти з використанням позиційних аргументів. Це дозволяє передавати значення параметрів у визначеному порядку. Необхідно переконатися у відповідності порядку аргументів сигнатурі конструктора.

На наступному етапі слід створити об’єкти з використанням іменованих аргументів. Це дає змогу явно вказувати відповідність між параметрами та значеннями, що підвищує наочність коду.

Після цього необхідно реалізувати перевірку параметрів у конструкторі. Наприклад, можна перевіряти діапазон значень або типи даних. У разі некоректних значень доцільно виводити повідомлення або використовувати механізми обробки помилок.

У процесі виконання роботи слід вивести значення атрибутів кожного об’єкта і переконатися, що вони ініціалізовані правильно. Доцільно використовувати окремий метод для відображення стану об’єкта.

Далі необхідно проаналізувати, як зміна параметрів впливає на поведінку об’єктів. Це дозволяє оцінити гнучкість реалізованого класу.

У разі виникнення помилок необхідно перевірити правильність передавання аргументів, відповідність їх кількості та правильність реалізації конструктора.

Після завершення роботи необхідно зробити висновки щодо особливостей використання конструкторів із параметрами.

Контрольні питання

1. Що таке конструктор із параметрами?
2. Яке призначення методу __init__?
3. Чим відрізняються позиційні та іменовані аргументи?
4. Як передаються параметри у конструктор?
5. Як ініціалізуються атрибути об’єкта?
6. Які помилки можуть виникати при передаванні параметрів?
7. Як перевірити коректність параметрів?
8. Чому важливо використовувати іменовані аргументи?
9. Як параметри впливають на стан об’єкта?
10. Які переваги мають параметризовані конструктори?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис реалізованого класу;
– перелік параметрів конструктора;
– приклади створених об’єктів;
– результати ініціалізації атрибутів;
– аналіз впливу параметрів;
– висновки.

Практична робота 9. Організація програмного коду з використанням модулів і пакетів Python

Мета роботи – набути практичних навичок організації програмного коду з використанням модулів і пакетів Python, навчитися розділяти програму на логічні частини та використовувати механізми імпорту для повторного використання коду.

Ключові положення

У процесі розробки програмного забезпечення важливим завданням є раціональна організація програмного коду. Якщо програма містить значну кількість функцій, класів і допоміжних елементів, її розміщення в одному файлі ускладнює читання, супровід і подальше розширення. Для розв'язання цієї проблеми у Python використовуються модулі і пакети.

Модуль у Python є окремим файлом із розширенням .py, який містить програмний код. У модулі можуть бути описані функції, класи, змінні та інші програмні об'єкти. Використання модулів дає змогу розділяти програму на самостійні частини відповідно до їх призначення. Наприклад, окремі модулі можуть містити класи предметної області, функції обробки даних або засоби взаємодії з користувачем.

Підключення модуля до програми виконується за допомогою оператора import. У Python підтримуються різні форми імпорту: імпорт цілого модуля, імпорт окремих об'єктів з модуля, а також використання псевдонімів. Це дозволяє гнучко організовувати доступ до програмних компонентів.

Пакет у Python є способом об'єднання модулів у спільну ієрархічну структуру. Пакет подається у вигляді каталогу, який містить модулі та, за потреби, вкладені каталоги. Така організація дає змогу логічно групувати модулі за функціональним призначенням і спрощує побудову великих програмних систем.

Використання модулів і пакетів сприяє повторному використанню програмного коду. Один і той самий модуль може бути підключений до різних програм, якщо він реалізує типові або універсальні функції. Це дозволяє уникати дублювання коду та підвищує ефективність розробки.

Важливим аспектом є правильне визначення меж між модулями. Кожен модуль повинен містити логічно пов'язані елементи. Надмірне дроблення коду ускладнює структуру програми, а надмірна концентрація великої кількості різнорідних елементів у одному модулі знижує її наочність.

Під час використання модулів і пакетів необхідно враховувати шляхи імпорту та структуру каталогів проєкту. Коректна організація файлів і каталогів забезпечує правильне підключення модулів та стабільну роботу програми.

Таким чином, використання модулів і пакетів є важливим засобом організації програмного коду у Python. Воно забезпечує структурованість, повторне використання коду та спрощує супровід програмних систем.

Практичне завдання

1. Створити програмний проєкт у Python.
2. Розділити програмний код на декілька модулів.

3. Розмістити у різних модулях класи, функції або змінні відповідно до їх призначення.

4. Реалізувати імпорт одного модуля в інший.

5. Використати різні форми оператора import.

6. Створити пакет для групування модулів.

7. Розмістити декілька модулів у структурі пакета.

8. Організувати використання класів або функцій з різних модулів пакета.

9. Перевірити працездатність програми після поділу коду на модулі і пакети.

10. Проаналізувати переваги такої організації програмного коду.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити програму або набір класів і функцій, код яких буде організовано за допомогою модулів і пакетів. Доцільно використовувати приклад, що містить декілька логічно відокремлених частин.

На першому етапі слід створити окремі файли модулів. У кожному модулі потрібно розмістити програмні об’єкти, які мають спільне призначення. Наприклад, один модуль може містити класи, інший – допоміжні функції, а третій – код основної програми.

Далі необхідно реалізувати імпорт модулів. Потрібно перевірити різні способи імпорту, зокрема підключення цілого модуля, імпорт окремих елементів і використання псевдонімів. Необхідно звернути увагу на те, як змінюється форма звертання до об’єктів при різних способах імпорту.

На наступному етапі слід створити пакет у вигляді окремого каталогу і розмістити у ньому декілька модулів. Доцільно згрупувати модулі за функціональним призначенням, щоб структура програми була логічною та зрозумілою.

Після цього потрібно підключити модулі з пакета у головну програму або в інші модулі. Необхідно переконатися, що імпорт виконується правильно, а програмні об’єкти доступні для використання.

У процесі виконання роботи слід перевірити працездатність усієї програми після її поділу на модулі і пакети. Особливу увагу потрібно приділити правильності шляхів імпорту та відсутності конфліктів імен.

У разі виникнення помилок необхідно перевірити структуру каталогів, правильність назв файлів і модулів, а також коректність використання операторів import. Доцільно також перевірити, чи всі необхідні елементи доступні з відповідних модулів.

Після завершення роботи необхідно проаналізувати, як використання модулів і пакетів вплинуло на структуру програми, її читабельність і можливість повторного використання коду.

Контрольні питання

1. Що таке модуль у Python?
2. Що таке пакет у Python?
3. Яке призначення оператора import?
4. Які існують форми імпорту у Python?
5. Які переваги дає використання модулів?
6. Для чого використовуються пакети?

7. Як організовується структура пакета?
8. Які помилки можуть виникати при імпорті модулів?
9. Чому важливо логічно розподіляти код між модулями?
10. Які переваги дає поділ програми на модулі і пакети?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис структури програмного проєкту;
– перелік створених модулів і пакетів;
– приклади використаних операторів імпорту;
– результати виконання програми;
– аналіз організації програмного коду;
– висновки.

ТЕМА 3. ІНКАПСУЛЯЦІЯ ТА ВЗАЄМОДІЯ ОБ'ЄКТІВ

Практична робота 10. Реалізація інкапсуляції у класах Python та організація доступу до атрибутів об'єкта

Мета роботи – набути практичних навичок реалізації інкапсуляції у класах Python, дослідити механізми обмеження доступу до атрибутів та організації контрольованого доступу до даних об'єкта.

Ключові положення

Інкапсуляція є одним із базових принципів об'єктно-орієнтованого програмування, який передбачає об'єднання даних і методів у межах одного класу та обмеження прямого доступу до внутрішнього стану об'єкта. Такий підхід забезпечує захист даних і підвищує надійність програмного забезпечення.

У Python інкапсуляція реалізується на основі домовленостей щодо іменування атрибутів. Атрибути з одним підкресленням на початку імені вважаються захищеними і призначені для внутрішнього використання. Атрибути з подвійним підкресленням мають механізм зміни імені, що ускладнює прямий доступ до них ззовні.

Прямий доступ до атрибутів об'єкта може призвести до неконтрольованої зміни їх значень. Для запобігання цьому використовуються методи доступу, які дозволяють читати або змінювати значення атрибутів із додатковою перевіркою. Такі методи забезпечують контроль за коректністю даних.

У Python для організації доступу до атрибутів широко використовуються властивості, що визначаються за допомогою декоратора @property. Це дозволяє реалізувати доступ до атрибутів як до звичайних змінних, але з можливістю виконання додаткових дій під час читання або зміни значення.

Інкапсуляція дозволяє відокремити внутрішню реалізацію класу від його зовнішнього інтерфейсу. Зовнішній код взаємодіє з об'єктом через методи або властивості, не маючи доступу до внутрішніх деталей реалізації.

Важливим аспектом є правильне визначення рівня доступу до атрибутів. Атрибути, що не повинні змінюватися безпосередньо, слід приховувати, а доступ до них організовувати через методи. Це дозволяє забезпечити цілісність даних.

Таким чином, інкапсуляція у Python є важливим механізмом організації програмного коду, який забезпечує захист даних, контроль доступу та підвищує якість програмного забезпечення.

Практичне завдання

1. Створити клас у мові Python.
2. Визначити атрибути об'єкта.
3. Реалізувати захищені атрибути з використанням одного підкреслення.
4. Реалізувати приховані атрибути з використанням подвійного підкреслення.
5. Реалізувати методи для доступу до атрибутів.
6. Використати декоратор @property для організації доступу.

7. Реалізувати перевірку значень при зміні атрибутів.
8. Створити об’єкти класу.
9. Перевірити обмеження доступу до атрибутів.
10. Проаналізувати результати.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно створити клас, який міститиме атрибути з різними рівнями доступу. Доцільно обрати приклад, у якому важливо контролювати зміну значень атрибутів, наприклад рахунок або студент.

На першому етапі слід визначити атрибути об’єкта. Частину з них необхідно реалізувати як відкриті, а частину – як захищені або приховані. Для цього використовуються відповідні правила іменування.

Далі потрібно реалізувати методи доступу до атрибутів. Це можуть бути методи для отримання значення атрибута або для його зміни. У методах зміни доцільно передбачити перевірку коректності значень.

На наступному етапі слід використати декоратор @property для організації доступу до атрибутів. Це дозволяє працювати з атрибутами як зі звичайними змінними, але з додатковим контролем.

Після цього необхідно створити об’єкти класу і перевірити доступ до їх атрибутів. Слід спробувати змінити значення атрибутів безпосередньо і через методи, щоб оцінити ефективність інкапсуляції.

У процесі виконання роботи необхідно звернути увагу на те, як інкапсуляція впливає на структуру програми і захист даних. Доцільно проаналізувати, які атрибути доцільно приховувати, а які можуть бути відкритими.

У разі виникнення помилок необхідно перевірити правильність використання декораторів, імен атрибутів і логіку перевірки значень.

Після завершення роботи необхідно зробити висновки щодо реалізації інкапсуляції у Python.

Контрольні питання

1. Що таке інкапсуляція?
2. Яка мета використання інкапсуляції?
3. Як реалізується інкапсуляція у Python?
4. Що означає використання одного підкреслення?
5. Що означає використання подвійного підкреслення?
6. Що таке методи доступу?
7. Яке призначення декоратора @property?
8. Як організувати перевірку значень атрибутів?
9. Чому важливо обмежувати доступ до даних?
10. Які переваги дає інкапсуляція?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис реалізованого класу;
– перелік атрибутів і способів доступу до них;
– приклади використання методів доступу;
– результати виконання програми;
– аналіз реалізації інкапсуляції;
– висновки.

Практична робота 11. Створення методів доступу до даних об’єкта та контроль зміни стану об’єкта

Мета роботи – набути практичних навичок реалізації методів доступу до атрибутів об’єкта, забезпечення контролю зміни стану об’єкта та перевірки коректності даних.

Ключові положення

У об’єктно-орієнтованому програмуванні важливим аспектом є контроль доступу до даних об’єкта. Прямий доступ до атрибутів може призвести до некоректних змін стану об’єкта, що порушує логіку роботи програми. Для запобігання цьому використовуються методи доступу, які забезпечують контрольовану взаємодію з атрибутами.

Методи доступу поділяються на методи отримання значень і методи зміни значень атрибутів. Перші дозволяють отримати значення атрибута без зміни стану об’єкта, другі – змінюють значення атрибута з урахуванням певних правил або обмежень.

У Python для реалізації методів доступу можуть використовуватися як звичайні методи, так і властивості, визначені за допомогою декоратора @property. Використання властивостей дозволяє звертатися до атрибутів як до звичайних змінних, але з можливістю виконання перевірок і додаткової логіки.

Контроль зміни стану об’єкта передбачає перевірку значень, які встановлюються атрибутам. Наприклад, можна перевіряти діапазон допустимих значень, типи даних або взаємозв’язок між атрибутами. Це дозволяє забезпечити коректність стану об’єкта протягом усього часу його існування.

Методи доступу також можуть використовуватися для обчислення значень на основі інших атрибутів. У такому випадку атрибут не зберігається безпосередньо, а обчислюється під час звертання до нього.

Важливим аспектом є забезпечення узгодженості стану об’єкта. Усі зміни атрибутів повинні виконуватися через визначені механізми доступу, що дозволяє уникнути неконтрольованих змін.

Таким чином, використання методів доступу та контроль зміни стану об’єкта є необхідними для забезпечення надійності та коректності роботи програм.

Практичне завдання

1. Створити клас у мові Python.
2. Визначити атрибути об’єкта.
3. Реалізувати методи отримання значень атрибутів.
4. Реалізувати методи зміни значень атрибутів.
5. Використати декоратор @property для доступу до атрибутів.
6. Реалізувати перевірку значень при зміні атрибутів.
7. Створити об’єкти класу.
8. Змінювати значення атрибутів через методи доступу.
9. Перевірити коректність зміни стану об’єкта.
10. Проаналізувати результати.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити клас, у якому буде реалізовано контроль доступу до даних. Доцільно обрати приклад, у якому важливо забезпечити коректність значень атрибутів, наприклад банківський рахунок або інформацію про студента.

На першому етапі слід визначити атрибути об’єкта та обрати, які з них повинні бути доступними безпосередньо, а які – лише через методи доступу. Атрибути, що потребують контролю, доцільно зробити захищеними або прихованими.

Далі необхідно реалізувати методи отримання значень атрибутів. Ці методи повинні повертати значення атрибутів без їх зміни.

На наступному етапі потрібно реалізувати методи зміни значень атрибутів. У цих методах необхідно передбачити перевірку коректності значень, наприклад перевірку діапазону або типу даних.

Після цього слід використати декоратор @property для організації доступу до атрибутів. Це дозволяє поєднати зручність використання і контроль за зміною даних.

У процесі виконання роботи необхідно створити об’єкти класу та виконати зміну значень атрибутів через реалізовані методи. Слід перевірити, що некоректні значення не встановлюються.

Особливу увагу потрібно приділити аналізу стану об’єкта після кожної зміни. Необхідно переконатися, що всі атрибути залишаються у допустимому стані.

У разі виникнення помилок необхідно перевірити правильність реалізації методів доступу, логіку перевірки значень і використання декоратора @property.

Після завершення роботи необхідно зробити висновки щодо ефективності використання методів доступу для контролю стану об’єкта.

Контрольні питання

1. Що таке методи доступу?
2. Які існують типи методів доступу?
3. Яке призначення методу отримання значення?
4. Яке призначення методу зміни значення?
5. Що таке декоратор @property?

6. Як організувати контроль зміни стану об’єкта?
7. Які перевірки можна виконувати при зміні атрибутів?
8. Чому не слід змінювати атрибути безпосередньо?
9. Як забезпечити узгодженість стану об’єкта?
10. Які переваги дає використання методів доступу?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис реалізованого класу;
– перелік методів доступу;
– приклади зміни значень атрибутів;
– результати виконання програми;
– аналіз контролю стану об’єкта;
– висновки.

Практична робота 11. Створення методів доступу до даних об’єкта та контроль зміни стану об’єкта

Мета роботи – набути практичних навичок реалізації методів доступу до атрибутів об’єкта, забезпечення контролю зміни стану об’єкта та перевірки коректності даних.

Ключові положення

У об’єктно-орієнтованому програмуванні важливим аспектом є контроль доступу до даних об’єкта. Прямий доступ до атрибутів може призвести до некоректних змін стану об’єкта, що порушує логіку роботи програми. Для запобігання цьому використовуються методи доступу, які забезпечують контрольовану взаємодію з атрибутами.

Методи доступу поділяються на методи отримання значень і методи зміни значень атрибутів. Перші дозволяють отримати значення атрибута без зміни стану об’єкта, другі – змінюють значення атрибута з урахуванням певних правил або обмежень.

У Python для реалізації методів доступу можуть використовуватися як звичайні методи, так і властивості, визначені за допомогою декоратора @property. Використання властивостей дозволяє звертатися до атрибутів як до звичайних змінних, але з можливістю виконання перевірок і додаткової логіки.

Контроль зміни стану об’єкта передбачає перевірку значень, які встановлюються атрибутам. Наприклад, можна перевіряти діапазон допустимих значень, типи даних або взаємозв’язок між атрибутами. Це дозволяє забезпечити коректність стану об’єкта протягом усього часу його існування.

Методи доступу також можуть використовуватися для обчислення значень на основі інших атрибутів. У такому випадку атрибут не зберігається безпосередньо, а обчислюється під час звертання до нього.

Важливим аспектом є забезпечення узгодженості стану об’єкта. Усі зміни атрибутів повинні виконуватися через визначені механізми доступу, що дозволяє уникнути неконтрольованих змін.

Таким чином, використання методів доступу та контроль зміни стану об’єкта є необхідними для забезпечення надійності та коректності роботи програм.

Практичне завдання

1. Створити клас у мові Python.
2. Визначити атрибути об’єкта.
3. Реалізувати методи отримання значень атрибутів.
4. Реалізувати методи зміни значень атрибутів.
5. Використати декоратор @property для доступу до атрибутів.
6. Реалізувати перевірку значень при зміні атрибутів.
7. Створити об’єкти класу.
8. Змінювати значення атрибутів через методи доступу.
9. Перевірити коректність зміни стану об’єкта.
10. Проаналізувати результати.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити клас, у якому буде реалізовано контроль доступу до даних. Доцільно обрати приклад, у якому важливо забезпечити коректність значень атрибутів, наприклад банківський рахунок або інформацію про студента.

На першому етапі слід визначити атрибути об’єкта та обрати, які з них повинні бути доступними безпосередньо, а які – лише через методи доступу. Атрибути, що потребують контролю, доцільно зробити захищеними або прихованими.

Далі необхідно реалізувати методи отримання значень атрибутів. Ці методи повинні повертати значення атрибутів без їх зміни.

На наступному етапі потрібно реалізувати методи зміни значень атрибутів. У цих методах необхідно передбачити перевірку коректності значень, наприклад перевірку діапазону або типу даних.

Після цього слід використати декоратор @property для організації доступу до атрибутів. Це дозволяє поєднати зручність використання і контроль за зміною даних.

У процесі виконання роботи необхідно створити об’єкти класу та виконати зміну значень атрибутів через реалізовані методи. Слід перевірити, що некоректні значення не встановлюються.

Особливу увагу потрібно приділити аналізу стану об’єкта після кожної зміни. Необхідно переконатися, що всі атрибути залишаються у допустимому стані.

У разі виникнення помилок необхідно перевірити правильність реалізації методів доступу, логіку перевірки значень і використання декоратора @property.

Після завершення роботи необхідно зробити висновки щодо ефективності використання методів доступу для контролю стану об’єкта.

Контрольні питання

1. Що таке методи доступу?
2. Які існують типи методів доступу?
3. Яке призначення методу отримання значення?
4. Яке призначення методу зміни значення?
5. Що таке декоратор @property?
6. Як організувати контроль зміни стану об’єкта?
7. Які перевірки можна виконувати при зміні атрибутів?
8. Чому не слід змінювати атрибути безпосередньо?
9. Як забезпечити узгодженість стану об’єкта?
10. Які переваги дає використання методів доступу?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис реалізованого класу;
– перелік методів доступу;
– приклади зміни значень атрибутів;
– результати виконання програми;
– аналіз контролю стану об’єкта;
– висновки.

Практична робота 12. Реалізація взаємодії між об’єктами різних класів у програмній системі

Мета роботи – набути практичних навичок реалізації взаємодії між об’єктами різних класів, організації обміну даними між ними та побудови узгодженої програмної системи.

Ключові положення

У об’єктно-орієнтованому програмуванні складні програмні системи будуються як сукупність взаємодіючих об’єктів, що належать до різних класів. Кожен клас описує окрему сутність предметної області, а взаємодія між об’єктами дозволяє реалізувати необхідну функціональність системи.

Взаємодія між об’єктами різних класів здійснюється через виклик методів і передачу об’єктів як параметрів. Один об’єкт може використовувати інший для виконання певних операцій або отримання даних. Це дозволяє розподілити відповідальність між класами та забезпечити модульність системи.

Одним із поширених підходів є використання композиції, коли об’єкт одного класу містить посилання на об’єкти інших класів. Такий підхід дозволяє будувати складні структури з простіших компонентів і забезпечує гнучкість у проєктуванні.

Іншим важливим аспектом є узгодження інтерфейсів взаємодії. Методи, які використовуються для взаємодії, повинні мати чітко визначені параметри і повертати передбачувані результати. Це дозволяє уникнути помилок і спростити використання класів.

Під час організації взаємодії між класами необхідно дотримуватися принципу розподілу відповідальності. Кожен клас повинен виконувати лише ті функції, які відповідають його призначенню. Надмірна залежність між класами ускладнює структуру системи та її супровід.

Важливим є також контроль за станом об’єктів під час взаємодії. Зміни, які відбуваються в одному об’єкті, можуть впливати на інші об’єкти, тому необхідно забезпечити узгодженість даних у системі.

Таким чином, реалізація взаємодії між об’єктами різних класів є основою побудови складних об’єктно-орієнтованих систем і вимагає ретельного проєктування структури класів і їх інтерфейсів.

Практичне завдання

1. Створити декілька взаємопов’язаних класів.
2. Визначити роль кожного класу у програмній системі.
3. Реалізувати атрибути і методи для кожного класу.
4. Організувати взаємодію між об’єктами різних класів.
5. Реалізувати передачу об’єктів як параметрів методів.
6. Використати композицію для об’єднання об’єктів.
7. Створити об’єкти кожного класу.
8. Змоделювати сценарій взаємодії між об’єктами.
9. Вивести результати роботи програми.
10. Проаналізувати коректність взаємодії.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити систему, яка містить декілька взаємодіючих сутностей. Доцільно використовувати приклади, у яких різні об’єкти виконують різні ролі, наприклад студент і курс, замовлення і товар, користувач і система.

На першому етапі слід визначити класи, які представляють основні сутності системи. Для кожного класу потрібно встановити його призначення, атрибути та методи.

Далі необхідно реалізувати зв’язки між класами. Це може бути зберігання посилань на об’єкти інших класів або передавання об’єктів як параметрів методів.

На наступному етапі слід створити об’єкти кожного класу та організувати їх взаємодію. Необхідно реалізувати сценарій, у якому об’єкти обмінюються даними або впливають один на одного.

У процесі виконання роботи необхідно перевірити, що взаємодія між об’єктами відбувається коректно, а дані передаються без помилок. Доцільно використовувати виведення результатів для контролю роботи програми.

Особливу увагу слід приділити узгодженості стану об’єктів. Після кожної операції необхідно перевіряти, чи відповідає стан об’єктів очікуваному результату.

У разі виникнення помилок необхідно перевірити правильність викликів методів, передачі параметрів та організацію зв’язків між класами.

Після завершення роботи необхідно проаналізувати структуру реалізованої системи та оцінити ефективність взаємодії між об’єктами.

Контрольні питання

1. Як організовується взаємодія між об’єктами різних класів?
2. Що таке композиція у об’єктно-орієнтованому програмуванні?
3. Яким чином передаються об’єкти між методами?
4. Що таке інтерфейс взаємодії?
5. Чому важливо розподіляти відповідальність між класами?
6. Як забезпечити узгодженість стану об’єктів?
7. Які помилки можуть виникати при взаємодії об’єктів?
8. Як перевірити коректність взаємодії?
9. Які переваги має використання декількох класів?
10. Як взаємодія об’єктів впливає на структуру програми?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис створених класів;
– опис взаємодії між об’єктами;
– приклади реалізації сценарію взаємодії;
– результати виконання програми;
– аналіз роботи системи;
– висновки.

Практична робота 13. Моделювання відносин між класами на основі асоціації, агрегації та композиції

Мета роботи – набути практичних навичок моделювання відносин між класами, дослідити особливості асоціації, агрегації та композиції та реалізувати їх у програмному коді.

Ключові положення

У об’єктно-орієнтованому програмуванні важливим аспектом є встановлення відносин між класами. Такі відносини визначають спосіб взаємодії об’єктів і впливають на структуру програмної системи. Найбільш поширеними видами відносин є асоціація, агрегація та композиція.

Асоціація є найзагальнішим видом зв’язку між класами. Вона відображає факт взаємодії об’єктів без жорсткої залежності між ними. Об’єкти можуть існувати незалежно один від одного, але використовують один одного для виконання певних дій. Такий зв’язок часто реалізується через передавання об’єктів як параметрів методів.

Агрегація є окремим випадком асоціації, який передбачає відношення частина–ціле. У цьому випадку один об’єкт містить інші об’єкти, але ці об’єкти можуть існувати незалежно від нього. Наприклад, група студентів містить студентів, але студенти можуть існувати окремо від групи.

Композиція є більш жорстким видом зв’язку, ніж агрегація. У цьому випадку об’єкти частин не можуть існувати без об’єкта-цілого. Видалення об’єкта-цілого призводить до знищення його складових частин. Це забезпечує більш тісний зв’язок між об’єктами.

Правильне визначення типу відносин між класами дозволяє створювати логічно узгоджені та гнучкі програмні системи. Невірно обраний тип зв’язку може призвести до ускладнення структури програми та труднощів у її супроводі.

Під час реалізації відносин між класами необхідно враховувати рівень залежності між об’єктами, спосіб їх створення та життєвий цикл. Це дозволяє правильно обрати тип зв’язку та забезпечити коректну взаємодію.

Таким чином, моделювання відносин між класами на основі асоціації, агрегації та композиції є важливим етапом проєктування об’єктно-орієнтованих програм.

Практичне завдання

1. Створити декілька класів, що представляють сутності предметної області.
2. Реалізувати асоціацію між класами.
3. Реалізувати агрегацію між класами.
4. Реалізувати композицію між класами.
5. Визначити атрибути і методи для кожного класу.
6. Створити об’єкти кожного класу.
7. Організувати взаємодію між об’єктами відповідно до типу зв’язку.
8. Змоделювати приклади використання кожного типу відносин.
9. Вивести результати роботи програми.
10. Проаналізувати відмінності між типами відносин.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити предметну область, у якій можна реалізувати різні типи відносин між класами. Доцільно обрати приклад, у якому є як незалежні об’єкти, так і об’єкти, що утворюють складні структури.

На першому етапі слід реалізувати асоціацію. Для цього необхідно створити класи, об’єкти яких взаємодіють, але не залежать один від одного. Взаємодія може бути реалізована через передавання об’єктів у методи.

Далі потрібно реалізувати агрегацію. Один із класів повинен містити посилання на об’єкти іншого класу, але ці об’єкти повинні створюватися незалежно. Необхідно перевірити, що об’єкти частин можуть існувати без об’єкта-цілого.

На наступному етапі слід реалізувати композицію. Об’єкти складових частин повинні створюватися всередині об’єкта-цілого і не існувати окремо від нього. Необхідно переконатися, що їх життєвий цикл залежить від об’єкта-цілого.

У процесі виконання роботи необхідно створити об’єкти і змоделювати їх взаємодію. Для кожного типу відносин слід реалізувати окремий приклад і проаналізувати його поведінку.

Особливу увагу потрібно приділити відмінностям між агрегацією та композицією. Необхідно чітко визначити, чи можуть об’єкти частин існувати незалежно.

У разі виникнення помилок необхідно перевірити правильність реалізації зв’язків між класами, способи створення об’єктів і логіку їх взаємодії.

Після завершення роботи необхідно проаналізувати отримані результати та зробити висновки щодо особливостей кожного типу відносин.

Контрольні питання

1. Що таке асоціація між класами?
2. Що таке агрегація?
3. Що таке композиція?
4. У чому різниця між агрегацією і композицією?
5. Як реалізується асоціація у програмному коді?
6. Як реалізується агрегація?
7. Як реалізується композиція?
8. Який вплив має життєвий цикл об’єктів на тип відносин?
9. Які помилки можуть виникати при моделюванні відносин?
10. Чому важливо правильно обирати тип зв’язку між класами?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис предметної області;
– опис реалізованих класів;
– приклади асоціації, агрегації та композиції;
– результати виконання програми;
– порівняльний аналіз типів відносин;
– висновки.

ТЕМА 4. УСПАДКУВАННЯ ТА ПОЛІМОРФІЗМ

Практична робота 14. Реалізація успадкування класів та побудова ієрархії класів у Python

Мета роботи – набути практичних навичок реалізації успадкування класів у Python, дослідити механізм створення похідних класів та побудови ієрархії класів у об'єктно-орієнтованій програмі.

Ключові положення

Успадкування є одним із базових принципів об'єктно-орієнтованого програмування, який дає змогу створювати нові класи на основі вже існуючих. Клас, від якого виконується успадкування, називається базовим, а клас, який створюється на його основі, – похідним. Такий підхід дозволяє повторно використовувати вже реалізовані атрибути і методи та розширювати функціональність програмного коду.

У Python успадкування реалізується шляхом указання імені базового класу в дужках під час оголошення похідного класу. Похідний клас автоматично одержує доступ до атрибутів і методів базового класу та може використовувати їх без повторного опису. Це спрощує побудову програм і зменшує дублювання коду.

Похідний клас може не лише використовувати успадковані елементи, а й доповнювати їх новими атрибутами та методами. Таким чином, він розширює функціональність базового класу відповідно до поставленого завдання. За потреби у похідному класі можуть бути реалізовані власні варіанти методів, які уточнюють або змінюють поведінку базового класу.

Побудова ієрархії класів передбачає створення кількох пов'язаних класів, які мають спільні властивості та відмінності. У верхній частині такої ієрархії розташовується більш загальний клас, а нижче – класи, що описують конкретніші сутності. Це дозволяє логічно організувати програмну систему та відобразити відношення загального і часткового між об'єктами предметної області.

Використання успадкування доцільне у випадках, коли кілька класів мають спільні характеристики або спільну поведінку. У такому разі загальні елементи доцільно винести до базового класу, а специфічні – залишити у похідних класах. Це сприяє підвищенню структурованості програми та полегшує її супровід.

Під час побудови ієрархії класів важливо забезпечити логічну узгодженість зв'язків між класами. Похідний клас повинен справді бути спеціалізованим різновидом базового класу. Якщо цей принцип не дотримується, структура програми стає штучною і менш зрозумілою.

Таким чином, успадкування є важливим засобом побудови об'єктно-орієнтованих програм у Python, який дає змогу реалізувати повторне використання коду та сформувати логічну ієрархію класів.

Практичне завдання

1. Створити базовий клас у мові Python.
2. Визначити атрибути і методи базового класу.
3. Створити один або декілька похідних класів.
4. Реалізувати успадкування атрибутів і методів базового класу.
5. Додати у похідні класи власні атрибути.
6. Додати у похідні класи власні методи.
7. Створити об’єкти базового і похідних класів.
8. Перевірити доступність успадкованих елементів у похідних класах.
9. Побудувати просту ієрархію класів для вибраної предметної області.
10. Проаналізувати результати роботи програми.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити предметну область, у якій доцільно використати ієрархію класів. Доцільно обрати приклад, у якому можна виділити загальну сутність і декілька її конкретних різновидів, наприклад транспортний засіб і його типи, працівник і категорії працівників, геометрична фігура і її окремі види.

На першому етапі слід створити базовий клас, який міститиме загальні атрибути і методи, характерні для всіх похідних класів. Необхідно визначити ті властивості, які будуть спільними для всіх об’єктів у межах вибраної ієрархії.

Далі потрібно створити похідні класи, які успадковують базовий клас. У цих класах необхідно реалізувати специфічні атрибути і методи, що відрізняють їх від базового класу. При цьому слід забезпечити збереження логічного зв’язку між загальним класом і його конкретнішими різновидами.

На наступному етапі необхідно створити об’єкти базового і похідних класів. Слід перевірити, що похідні класи мають доступ до успадкованих атрибутів і методів, а також коректно використовують власні елементи.

У процесі виконання роботи доцільно побудувати просту ієрархію класів із двох або трьох рівнів. Це дозволить краще простежити механізм передачі властивостей від базового класу до похідних. Необхідно звернути увагу на те, які атрибути і методи доцільно розміщувати у базовому класі, а які – у похідних.

Особливу увагу слід приділити перевірці роботи успадкованих методів. Потрібно переконатися, що об’єкти похідних класів можуть використовувати методи базового класу без повторної реалізації, якщо це відповідає логіці програми.

У разі виникнення помилок необхідно перевірити правильність оголошення похідних класів, коректність створення об’єктів та відповідність структури ієрархії змісту предметної області.

Після завершення роботи необхідно проаналізувати отримані результати та зробити висновки щодо доцільності використання успадкування і побудови ієрархії класів у Python.

Контрольні питання

1. Що таке успадкування у об’єктно-орієнтованому програмуванні?
2. Який клас називається базовим?

3. Який клас називається похідним?
4. Як реалізується успадкування у Python?
5. Які елементи базового класу успадковує похідний клас?
6. У чому полягає побудова ієрархії класів?
7. Які переваги дає використання успадкування?
8. Які вимоги висуваються до логічної побудови ієрархії класів?
9. Які помилки можуть виникати при реалізації успадкування?
10. У яких випадках доцільно використовувати успадкування?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис базового і похідних класів;
– схема або опис побудованої ієрархії класів;
– приклади створених об’єктів;
– результати перевірки успадкування атрибутів і методів;
– аналіз роботи програми;
– висновки.

Практична робота 15. Дослідження механізму розширення функціональності похідних класів

Мета роботи – набути практичних навичок розширення функціональності похідних класів у Python, дослідити способи доповнення успадкованих класів новими атрибутами і методами та проаналізувати особливості їх використання у програмі.

Ключові положення

У об’єктно-орієнтованому програмуванні похідний клас не лише успадковує властивості базового класу, а й може бути розширений відповідно до потреб конкретної задачі. Це означає, що у похідному класі можна додавати нові атрибути та методи, які відсутні у базовому класі, але необхідні для опису спеціалізованої поведінки об’єкта.

Розширення функціональності похідного класу є природним продовженням механізму успадкування. Базовий клас містить загальні елементи, спільні для групи об’єктів, а похідний клас додає специфічні можливості, які властиві лише окремому різновиду цих об’єктів. Такий підхід дозволяє поєднати повторне використання коду та спеціалізацію поведінки.

Під час розширення похідного класу нові атрибути зазвичай використовуються для опису додаткових характеристик об’єкта. Наприклад, якщо базовий клас описує загальний транспортний засіб, то похідний клас може містити додаткові дані, що характеризують конкретний його вид. Аналогічно нові методи реалізують дії, які не передбачені у базовому класі.

Розширення функціональності не повинно порушувати логіку побудованої ієрархії класів. Похідний клас має залишатися спеціалізованим різновидом базового класу, а додані елементи повинні відповідати його призначенню. Якщо у похідному класі з’являються

елементи, що не мають логічного зв’язку з базовим класом, структура програми стає менш узгодженою.

Важливим аспектом є правильна організація конструктора похідного класу. Якщо похідний клас містить додаткові атрибути, їх необхідно ініціалізувати під час створення об’єкта. При цьому слід забезпечити також коректну ініціалізацію успадкованих атрибутів базового класу.

Розширення функціональності похідних класів дозволяє створювати гнучкі та масштабовані програмні системи. За рахунок цього механізму одна загальна основа може бути використана для побудови кількох спеціалізованих класів із різними можливостями.

Таким чином, дослідження механізму розширення функціональності похідних класів є важливим етапом засвоєння об’єктно-орієнтованого підходу до розробки програмного забезпечення у Python.

Практичне завдання

1. Створити базовий клас у мові Python.
2. Створити один або декілька похідних класів.
3. Реалізувати успадкування атрибутів і методів базового класу.
4. Додати у похідний клас нові атрибути.
5. Додати у похідний клас нові методи.
6. Реалізувати конструктор похідного класу з урахуванням нових атрибутів.
7. Створити об’єкти похідного класу.
8. Перевірити використання успадкованих і нових методів.
9. Порівняти функціональність базового і похідного класів.
10. Проаналізувати результати роботи програми.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити предметну область, у якій доцільно використати базовий і похідний класи. Доцільно обрати приклад, у якому загальний клас може бути доповнений новими характеристиками і діями у похідному класі.

На першому етапі слід створити базовий клас, який міститиме загальні атрибути і методи. Потрібно визначити такі елементи, які є спільними для всіх об’єктів вибраної групи.

Далі необхідно створити похідний клас, який успадковує базовий. У цьому класі потрібно додати нові атрибути, що описують додаткові характеристики об’єкта, а також нові методи, які реалізують його спеціалізовану поведінку.

На наступному етапі слід реалізувати конструктор похідного класу. У ньому необхідно забезпечити ініціалізацію як успадкованих, так і нових атрибутів. Потрібно переконатися, що під час створення об’єкта всі його характеристики набувають коректних значень.

Після цього необхідно створити об’єкти базового і похідного класів. Слід перевірити, що об’єкти похідного класу можуть використовувати як успадковані методи, так і нові методи, визначені безпосередньо у похідному класі.

У процесі виконання роботи доцільно порівняти можливості базового і похідного класів. Необхідно визначити, які саме нові елементи розширюють функціональність похідного класу та як це впливає на поведінку об’єктів.

Особливу увагу слід приділити логічній узгодженості структури класів. Усі нові атрибути і методи повинні відповідати змісту похідного класу та не суперечити призначенню базового класу.

У разі виникнення помилок необхідно перевірити правильність реалізації успадкування, коректність конструктора похідного класу та відповідність нових елементів структурі класу.

Після завершення роботи необхідно проаналізувати результати та зробити висновки щодо способів розширення функціональності похідних класів у Python.

Контрольні питання

1. Яким чином похідний клас може розширювати функціональність базового класу?
2. Які елементи можна додавати у похідний клас?
3. Чому розширення функціональності є продовженням механізму успадкування?
4. Які вимоги висуваються до нових атрибутів і методів похідного класу?
5. Як організовується конструктор похідного класу?
6. Чим відрізняється функціональність базового і похідного класів?
7. Як перевірити коректність розширення класу?
8. Які помилки можуть виникати при додаванні нових елементів у похідний клас?
9. Чому важливо забезпечувати логічну узгодженість ієрархії класів?
10. Які переваги дає розширення функціональності похідних класів?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис базового і похідного класів;
– перелік доданих атрибутів і методів;
– приклади створених об’єктів;
– результати перевірки роботи успадкованих і нових елементів;
– аналіз розширення функціональності похідного класу;
– висновки.

Практична робота 16. Реалізація перевизначення методів у похідних класах

Мета роботи – набути практичних навичок реалізації перевизначення методів у похідних класах, дослідити особливості зміни поведінки успадкованих методів та проаналізувати їх використання у програмі.

Ключові положення

У об’єктно-орієнтованому програмуванні похідний клас може не лише успадковувати атрибути і методи базового класу, а й змінювати поведінку успадкованих методів відповідно до власного призначення. Така можливість називається перевизначенням методів. Вона

дозволяє використовувати спільну структуру базового класу, але реалізовувати іншу поведінку для конкретніших типів об’єктів.

Перевизначення методу виконується шляхом створення у похідному класі методу з таким самим іменем, як у базовому класі. Якщо для об’єкта похідного класу викликається цей метод, використовується саме реалізація похідного класу. У результаті похідний клас може уточнювати або повністю змінювати поведінку, визначену у базовому класі.

Такий механізм є важливим для побудови гнучких ієрархій класів. Базовий клас описує загальну поведінку, спільну для всіх об’єктів певної групи, а похідні класи уточнюють цю поведінку відповідно до своїх особливостей. Це дозволяє зберегти загальну логіку програми і водночас реалізувати відмінності між окремими типами об’єктів.

Перевизначення методів повинно бути логічно обґрунтованим. Похідний клас має змінювати лише ті методи, поведінка яких дійсно потребує уточнення. Якщо перевизначення виконується без необхідності, структура програми ускладнюється, а повторне використання коду зменшується.

У процесі перевизначення важливо враховувати, що інтерфейс методу у похідному класі повинен залишатися узгодженим із методом базового класу. Це означає, що його призначення, склад параметрів і загальна роль у системі повинні залишатися зрозумілими і передбачуваними. Такий підхід забезпечує коректну роботу ієрархії класів.

Перевизначення методів безпосередньо пов’язане з поліморфною поведінкою об’єктів. Один і той самий виклик методу може приводити до різних результатів залежно від того, об’єкт якого класу використовується. Завдяки цьому програмна система набуває гнучкості та кращої пристосованості до розширення.

Таким чином, перевизначення методів у похідних класах є важливим механізмом об’єктно-орієнтованого програмування, що дозволяє уточнювати поведінку об’єктів і будувати більш гнучкі програмні системи.

Практичне завдання

1. Створити базовий клас у мові Python.
2. Реалізувати у базовому класі один або декілька методів.
3. Створити похідний клас на основі базового.
4. Реалізувати у похідному класі метод з таким самим іменем, як у базовому класі.
5. Перевизначити поведінку успадкованого методу.
6. Створити об’єкти базового і похідного класів.
7. Викликати однойменні методи для об’єктів різних класів.
8. Порівняти результати виконання методів базового і похідного класів.
9. Проаналізувати доцільність перевизначення методу у вибраному прикладі.
10. Зробити висновки щодо особливостей перевизначення методів.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити предметну область, у якій доцільно використати базовий і похідний класи з різною поведінкою методів. Доцільно обрати приклад, у якому загальна дія є спільною для кількох об’єктів, але її конкретна реалізація відрізняється. Наприклад, це можуть бути геометричні фігури, транспортні засоби, працівники або навчальні об’єкти.

На першому етапі слід створити базовий клас і реалізувати у ньому метод, що описує загальну поведінку. Цей метод повинен мати зміст, спільний для всіх об’єктів, що належать до побудованої ієрархії.

Далі необхідно створити похідний клас і реалізувати у ньому метод з таким самим іменем. У цьому методі потрібно змінити або уточнити поведінку, визначену у базовому класі. Необхідно забезпечити, щоб нова реалізація відповідала логіці похідного класу.

На наступному етапі потрібно створити об’єкти базового і похідного класів та виконати виклик перевизначеного методу для кожного з них. Це дозволить порівняти результати роботи і простежити, як змінюється поведінка методу залежно від типу об’єкта.

У процесі виконання роботи доцільно звернути увагу на однаковість імені методу та логічну узгодженість його призначення у базовому і похідному класах. Перевизначення не повинно порушувати загальну структуру ієрархії.

Особливу увагу слід приділити аналізу відмінностей між успадкованим і перевизначеним методом. Необхідно встановити, які саме зміни внесені у похідному класі і як вони впливають на функціональність програми.

У разі виникнення помилок необхідно перевірити правильність оголошення похідного класу, збіг імен методів, кількість параметрів і коректність викликів методів для об’єктів різних класів.

Після завершення роботи необхідно проаналізувати результати та зробити висновки щодо доцільності та особливостей перевизначення методів у похідних класах.

Контрольні питання

1. Що таке перевизначення методу?
2. У якому випадку використовується перевизначення методів?
3. Як реалізується перевизначення методу у Python?
4. Яка умова повинна виконуватися для перевизначення методу у похідному класі?
5. Чим відрізняється успадкований метод від перевизначеного?
6. Як перевизначення методів пов’язане з успадкуванням?
7. Чому перевизначення методів сприяє гнучкості програми?
8. Які помилки можуть виникати при перевизначенні методів?
9. Чому важливо зберігати логічну узгодженість методу у базовому і похідному класах?
10. Які переваги дає перевизначення методів у об’єктно-орієнтованому програмуванні?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис базового і похідного класів;
– опис методу базового класу та його перевизначення;
– приклади створених об’єктів;
– результати виконання програми;
– аналіз відмінностей між реалізаціями методу;
– висновки.

Практична робота 17. Дослідження механізму поліморфізму під час виконання програм

Мета роботи – набути практичних навичок використання поліморфізму у Python, дослідити механізм динамічного зв'язування методів та проаналізувати поведінку об'єктів різних класів під час виконання програм.

Ключові положення

Поліморфізм є одним із ключових принципів об'єктно-орієнтованого програмування, який забезпечує можливість використання єдиного інтерфейсу для об'єктів різних типів. Це означає, що одна і та сама операція може виконуватися по-різному залежно від типу об'єкта, для якого вона викликається.

У Python поліморфізм реалізується на основі динамічного зв'язування методів. Це означає, що вибір конкретної реалізації методу відбувається під час виконання програми, а не на етапі її компіляції. У результаті виклик методу визначається типом об'єкта, а не типом змінної, що на нього посилається.

Поліморфізм тісно пов'язаний із механізмом успадкування. Похідні класи можуть перевизначати методи базового класу, і під час виклику методу використовується відповідна реалізація залежно від конкретного об'єкта. Це дозволяє реалізувати різну поведінку для об'єктів, що мають спільний інтерфейс.

Важливим аспектом є використання однакових імен методів у різних класах. Це забезпечує єдність інтерфейсу і дозволяє працювати з об'єктами різних типів у однаковий спосіб. Наприклад, можна створити список об'єктів різних класів і викликати для кожного з них один і той самий метод.

Поліморфізм дозволяє зменшити залежність між компонентами програми. Замість перевірки типу об'єкта можна використовувати єдиний інтерфейс, що спрощує код і підвищує його гнучкість.

Дослідження механізму поліморфізму передбачає аналіз того, як різні об'єкти реагують на однакові виклики методів і як змінюється поведінка програми залежно від типу об'єкта.

Таким чином, поліморфізм є важливим засобом побудови гнучких і розширюваних програмних систем у Python.

Практичне завдання

1. Створити базовий клас із методом.
2. Створити декілька похідних класів.
3. Перевизначити метод у похідних класах.
4. Створити об'єкти різних класів.
5. Об'єднати об'єкти у спільну колекцію.
6. Викликати один і той самий метод для всіх об'єктів.
7. Проаналізувати результати виконання методів.
8. Дослідити вплив типу об'єкта на результат виклику.
9. Порівняти поведінку об'єктів різних класів.
10. Зробити висновки щодо механізму поліморфізму.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити базовий клас, який міститиме метод, спільний для всіх похідних класів. Доцільно обрати приклад, у якому одна дія може виконуватися по-різному залежно від типу об’єкта, наприклад обчислення характеристик об’єкта, відображення інформації або виконання певної операції.

На першому етапі слід створити базовий клас і реалізувати у ньому метод, що визначає загальну поведінку. Далі необхідно створити декілька похідних класів, у яких цей метод буде перевизначений відповідно до специфіки кожного класу.

Після цього потрібно створити об’єкти різних класів і об’єднати їх у спільну структуру, наприклад список. Це дозволить працювати з ними як із єдиною групою.

На наступному етапі необхідно виконати виклик одного і того самого методу для кожного об’єкта у колекції. Потрібно проаналізувати, як змінюється результат виконання залежно від типу об’єкта.

У процесі виконання роботи слід звернути увагу на механізм динамічного зв’язування. Необхідно переконатися, що викликається саме та реалізація методу, яка відповідає типу об’єкта.

Особливу увагу слід приділити відсутності необхідності перевірки типу об’єкта. Поліморфізм дозволяє працювати з об’єктами через спільний інтерфейс без додаткових умов.

У разі виникнення помилок необхідно перевірити правильність перевизначення методів, коректність створення об’єктів та їх включення у колекцію.

Після завершення роботи необхідно проаналізувати результати та зробити висновки щодо особливостей реалізації поліморфізму у Python.

Контрольні питання

1. Що таке поліморфізм?
2. Яка роль поліморфізму у об’єктно-орієнтованому програмуванні?
3. Як реалізується поліморфізм у Python?
4. Що таке динамічне зв’язування методів?
5. Як пов’язані поліморфізм і успадкування?
6. Чому важливо використовувати єдиний інтерфейс?
7. Як поводяться об’єкти різних класів при однаковому виклику методу?
8. Чому не потрібно перевіряти тип об’єкта при використанні поліморфізму?
9. Які помилки можуть виникати при реалізації поліморфізму?
10. Які переваги дає використання поліморфізму?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис базового і похідних класів;
– опис перевизначених методів;
– приклади створених об’єктів;
– результати виконання програми;
– висновки.

ТЕМА 5. АБСТРАКЦІЯ ТА КОМПОНЕНТНА ОРГАНІЗАЦІЯ ПРОГРАМНИХ СИСТЕМ

Практична робота 18. Реалізація абстрактних класів та абстрактних методів у Python

Мета роботи – набути практичних навичок реалізації абстрактних класів і абстрактних методів у Python, дослідити їх призначення та особливості використання під час побудови ієрархії класів.

Ключові положення

У об'єктно-орієнтованому програмуванні не всі класи призначені для безпосереднього створення об'єктів. У багатьох випадках доцільно використовувати класи, які задають лише загальну структуру і спільний інтерфейс для групи похідних класів. Такі класи називаються абстрактними.

Абстрактний клас використовується для опису спільних властивостей і поведінки, які повинні бути характерними для декількох конкретних класів. Він може містити як звичайні методи з реалізованою поведінкою, так і абстрактні методи, для яких задається лише оголошення без реалізації. Конкретна реалізація таких методів виконується у похідних класах.

У Python для створення абстрактних класів використовується модуль abc. Абстрактний клас зазвичай успадковується від базового класу ABC, а абстрактні методи позначаються за допомогою декоратора @abstractmethod. Якщо у класі є хоча б один абстрактний метод, створення його об'єктів безпосередньо не допускається.

Абстрактний метод визначає обов'язкову поведінку, яку повинні реалізувати всі похідні класи. Це дозволяє забезпечити єдність інтерфейсу у межах ієрархії класів. Кожен похідний клас зобов'язаний надати власну реалізацію такого методу, інакше він також вважатиметься абстрактним.

Використання абстрактних класів сприяє кращій організації програмного коду. Вони дозволяють чітко відокремити спільні елементи від специфічних реалізацій і забезпечують логічну основу для побудови ієрархії класів. Це особливо важливо у випадках, коли декілька класів повинні мати однаковий набір методів, але реалізовувати їх по-різному.

Абстрактні класи тісно пов'язані з механізмами успадкування і поліморфізму. Вони задають загальний інтерфейс, а похідні класи забезпечують конкретну реалізацію. Завдяки цьому програма може працювати з об'єктами різних класів через спільний набір методів.

Таким чином, абстрактні класи та абстрактні методи є важливими засобами проєктування об'єктно-орієнтованих програм у Python, які дозволяють формалізувати спільний інтерфейс і забезпечити узгодженість структури класів.

Практичне завдання

1. Підключити модуль abc у програмі Python.
2. Створити абстрактний клас.
3. Реалізувати у ньому один або декілька абстрактних методів.

4. Додати до абстрактного класу звичайні методи або атрибути.
5. Створити один або декілька похідних класів.
6. Реалізувати у похідних класах абстрактні методи.
7. Створити об’єкти похідних класів.
8. Викликати реалізовані методи для об’єктів різних класів.
9. Перевірити неможливість створення об’єкта абстрактного класу.
10. Проаналізувати результати виконання програми.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити предметну область, у якій декілька класів повинні мати спільний інтерфейс, але різну реалізацію окремих методів. Доцільно обрати приклад, у якому існує загальна сутність і декілька її конкретних різновидів, наприклад геометрична фігура, транспортний засіб, працівник або навчальний об’єкт.

На першому етапі слід підключити модуль abc і створити абстрактний клас. Для цього необхідно успадкувати клас від ABC. У межах абстрактного класу потрібно визначити спільні атрибути або звичайні методи, які будуть однаковими для всіх похідних класів.

Далі необхідно оголосити один або декілька абстрактних методів за допомогою декоратора @abstractmethod. У таких методах реалізація не задається, а лише визначається їх призначення у межах спільного інтерфейсу.

На наступному етапі потрібно створити похідні класи, які успадковують абстрактний клас. У кожному з них необхідно реалізувати всі абстрактні методи. Реалізація повинна відповідати логіці конкретного класу і водночас зберігати загальне призначення методу, визначене в абстрактному класі.

Після цього слід створити об’єкти похідних класів і перевірити їх роботу. Необхідно виконати виклик реалізованих методів і переконатися, що кожен клас забезпечує власну поведінку в межах спільного інтерфейсу.

Окремо доцільно перевірити, що створення об’єкта абстрактного класу є неможливим. Це дозволяє наочно підтвердити особливість абстрактних класів як засобу задання структури, а не безпосереднього створення об’єктів.

У процесі виконання роботи слід звернути увагу на зв’язок абстрактних класів із поліморфізмом. Необхідно проаналізувати, як один і той самий виклик методу приводить до різної поведінки об’єктів різних похідних класів.

У разі виникнення помилок необхідно перевірити правильність підключення модуля abc, наявність декоратора @abstractmethod, а також повноту реалізації абстрактних методів у похідних класах.

Після завершення роботи необхідно проаналізувати отримані результати та зробити висновки щодо доцільності використання абстрактних класів і абстрактних методів у Python.

Контрольні питання

1. Що таке абстрактний клас?
2. Яке призначення абстрактного класу?
3. Що таке абстрактний метод?
4. Який модуль використовується у Python для реалізації абстрактних класів?

5. Яке призначення класу ABC?
6. Для чого використовується декоратор @abstractmethod?
7. Чи можна створити об’єкт абстрактного класу?
8. Які вимоги висуваються до похідного класу, що успадковує абстрактний клас?
9. Як абстрактні класи пов’язані з поліморфізмом?
10. Які переваги дає використання абстрактних класів у програмуванні?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис абстрактного класу;
– перелік абстрактних і звичайних методів;
– опис похідних класів;
– приклади створених об’єктів;
– результати виконання програми;
– аналіз використання абстрактних класів і методів;
– висновки.

Практична робота 19. Реалізація інтерфейсного підходу до проєктування програмних систем

Мета роботи – набути практичних навичок застосування інтерфейсного підходу до проєктування програмних систем у Python, дослідити роль інтерфейсів у забезпеченні гнучкості та розширюваності програмного забезпечення.

Ключові положення

Інтерфейсний підхід до проєктування програмних систем передбачає відокремлення опису поведінки об’єктів від їх конкретної реалізації. Інтерфейс визначає набір методів, які повинен реалізувати клас, але не містить їх конкретної реалізації. Це дозволяє різним класам реалізовувати однаковий інтерфейс різними способами.

У Python інтерфейси зазвичай реалізуються за допомогою абстрактних класів. Абстрактний клас визначає набір абстрактних методів, які повинні бути реалізовані у похідних класах. Таким чином, він виконує роль інтерфейсу, забезпечуючи єдність взаємодії між компонентами програмної системи.

Використання інтерфейсів дозволяє зменшити залежність між частинами програми. Замість роботи з конкретними класами програма взаємодіє з об’єктами через визначений інтерфейс. Це спрощує зміну реалізації без впливу на інші частини системи.

Інтерфейсний підхід забезпечує розширюваність програмного забезпечення. Нові класи можуть бути додані до системи за умови реалізації визначеного інтерфейсу, що дозволяє розширювати функціональність без зміни існуючого коду.

Важливим аспектом є узгодженість інтерфейсу. Усі класи, що реалізують інтерфейс, повинні забезпечувати однаковий набір методів і дотримуватися їх призначення. Це дозволяє використовувати об’єкти різних класів взаємозамінно.

Інтерфейсний підхід тісно пов’язаний із поліморфізмом. Один і той самий виклик методу може виконуватися різними способами залежно від реалізації у конкретному класі. Це дозволяє будувати гнучкі програмні системи.

Таким чином, інтерфейсний підхід є важливим засобом проєктування програмного забезпечення, що забезпечує модульність, гнучкість і можливість розширення системи.

Практичне завдання

1. Визначити інтерфейс у вигляді абстрактного класу.
2. Оголосити у ньому один або декілька абстрактних методів.
3. Створити декілька класів, що реалізують цей інтерфейс.
4. Реалізувати всі абстрактні методи у похідних класах.
5. Створити об’єкти різних класів, що реалізують інтерфейс.
6. Організувати використання об’єктів через спільний інтерфейс.
7. Реалізувати сценарій взаємодії об’єктів у програмі.
8. Продемонструвати поліморфну поведінку об’єктів.
9. Проаналізувати можливість заміни одного класу іншим.
10. Зробити висновки щодо ефективності інтерфейсного підходу.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити задачу, у якій декілька класів повинні виконувати однакові дії, але різними способами. Доцільно обрати приклад, у якому можна виділити спільний набір операцій для різних типів об’єктів.

На першому етапі слід створити абстрактний клас, який виконуватиме роль інтерфейсу. У ньому необхідно оголосити абстрактні методи, що визначають спільну поведінку для всіх класів, які будуть його реалізовувати.

Далі потрібно створити декілька похідних класів, які реалізують цей інтерфейс. У кожному класі необхідно надати власну реалізацію абстрактних методів відповідно до логіки конкретного об’єкта.

На наступному етапі слід створити об’єкти різних класів і організувати їх використання через спільний інтерфейс. Необхідно забезпечити, щоб виклики методів виконувалися однаково з точки зору користувача програми.

У процесі виконання роботи доцільно продемонструвати поліморфну поведінку об’єктів. Для цього можна використати колекцію об’єктів різних класів і виконати для них однакові виклики методів.

Особливу увагу слід приділити можливості заміни одного класу іншим. Необхідно переконатися, що зміна конкретної реалізації не потребує зміни коду, який використовує інтерфейс.

У разі виникнення помилок необхідно перевірити правильність реалізації абстрактних методів, відповідність інтерфейсу та узгодженість імен методів у різних класах.

Після завершення роботи необхідно проаналізувати результати та зробити висновки щодо переваг інтерфейсного підходу у проєктуванні програмних систем.

Контрольні питання

1. Що таке інтерфейс у програмуванні?
2. Як реалізуються інтерфейси у Python?
3. Яке призначення абстрактного класу у ролі інтерфейсу?
4. Чому інтерфейсний підхід зменшує залежність між компонентами?
5. Як інтерфейси пов’язані з поліморфізмом?
6. Які вимоги висуваються до класів, що реалізують інтерфейс?
7. Чому важливо забезпечувати узгодженість інтерфейсу?
8. Як інтерфейс сприяє розширюваності програмної системи?
9. Які помилки можуть виникати при реалізації інтерфейсів?
10. Які переваги має інтерфейсний підхід?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис інтерфейсу;
– перелік абстрактних методів;
– опис класів, що реалізують інтерфейс;
– приклади створених об’єктів;
– результати виконання програми;
– аналіз поліморфної поведінки;
– висновки.

Практична робота 20. Реалізація абстракції та компонентної організації програмних систем

Мета роботи – набути практичних навичок застосування принципу абстракції та компонентного підходу до побудови програмних систем, дослідити розподіл функціональності між компонентами та організацію їх взаємодії.

Ключові положення

Абстракція є фундаментальним принципом об’єктно-орієнтованого програмування, який передбачає виділення суттєвих характеристик об’єкта та приховування другорядних деталей реалізації. Завдяки абстракції розробник працює не з конкретною реалізацією, а з узагальненою моделлю, що описує поведінку системи.

У програмних системах абстракція дозволяє визначити рівні представлення. На верхньому рівні задається загальна логіка та інтерфейси взаємодії, а на нижчих рівнях реалізуються конкретні деталі. Такий підхід забезпечує незалежність компонентів і спрощує зміну реалізації без впливу на інші частини системи.

Компонентна організація передбачає поділ програмної системи на окремі логічно завершені частини – компоненти. Кожен компонент виконує чітко визначену функцію і

взаємодіє з іншими компонентами через інтерфейси. Це дозволяє будувати складні системи з незалежних модулів.

Компоненти повинні бути слабко зв'язаними та мати чітко визначені межі відповідальності. Слабке зв'язування означає, що зміни в одному компоненті не повинні вимагати змін в інших. Це досягається завдяки використанню абстрактних інтерфейсів і узгоджених способів взаємодії.

Інтерфейси відіграють ключову роль у компонентній архітектурі. Вони визначають, які операції доступні для взаємодії з компонентом, не розкриваючи внутрішню реалізацію. Це дозволяє замінювати компоненти без зміни логіки системи.

Використання абстракції та компонентного підходу дозволяє підвищити масштабованість програмних систем. Нові компоненти можуть бути додані без зміни існуючих, якщо вони дотримуються визначених інтерфейсів.

Таким чином, абстракція та компонентна організація є основою побудови складних програмних систем, що забезпечує їх гнучкість, модульність і можливість розширення.

Практичне завдання

1. Визначити загальну структуру програмної системи.
2. Виділити основні компоненти системи.
3. Визначити відповідальність кожного компонента.
4. Реалізувати абстрактні інтерфейси для взаємодії між компонентами.
5. Створити класи, що реалізують окремі компоненти.
6. Організувати взаємодію компонентів через інтерфейси.
7. Забезпечити незалежність компонентів один від одного.
8. Реалізувати простий сценарій роботи системи.
9. Перевірити можливість заміни одного компонента іншим.
10. Проаналізувати результати.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно визначити просту програмну систему, яка складається з декількох компонентів. Доцільно обрати приклад, у якому можна виділити окремі функціональні частини, наприклад систему обробки даних, інформаційну систему або модель взаємодії користувача з програмою.

На першому етапі слід визначити компоненти системи. Необхідно встановити, які частини програми виконують окремі функції, і сформулювати їх призначення. Кожен компонент повинен мати чітко визначену відповідальність.

Далі потрібно визначити інтерфейси взаємодії між компонентами. Для цього доцільно використати абстрактні класи, які задають набір методів для взаємодії. Інтерфейси повинні бути узгодженими і достатніми для виконання поставлених задач.

На наступному етапі необхідно реалізувати конкретні класи компонентів. Кожен клас повинен реалізовувати визначений інтерфейс і містити власну логіку роботи. При цьому внутрішня реалізація компонентів повинна бути прихована від інших частин системи.

Після цього слід організувати взаємодію компонентів. Необхідно забезпечити, щоб компоненти обмінювалися даними лише через визначені інтерфейси. Прямий доступ до внутрішніх даних інших компонентів не допускається.

У процесі виконання роботи необхідно перевірити незалежність компонентів. Для цього доцільно змінити реалізацію одного з компонентів і переконатися, що це не впливає на інші частини системи.

Особливу увагу слід приділити узгодженості інтерфейсів і правильності розподілу відповідальності між компонентами. Кожен компонент повинен виконувати лише свою функцію.

У разі виникнення помилок необхідно перевірити правильність реалізації інтерфейсів, взаємодії компонентів і відповідність структури системи поставленому завданню.

Після завершення роботи необхідно проаналізувати отримані результати та зробити висновки щодо ефективності використання абстракції та компонентного підходу.

Контрольні питання

1. Що таке абстракція у програмуванні?
2. Яке призначення абстракції?
3. Що таке компонент у програмній системі?
4. Які переваги має компонентна організація?
5. Що означає слабке зв’язування компонентів?
6. Яку роль відіграють інтерфейси у компонентній архітектурі?
7. Як забезпечити незалежність компонентів?
8. Чому важливо розподіляти відповідальність між компонентами?
9. Які помилки можуть виникати при компонентному проєктуванні?
10. Які переваги дає використання абстракції та компонентного підходу?

Зміст звіту

– назва практичної роботи;
– мета виконання;
– опис структури програмної системи;
– перелік компонентів і їх призначення;
– опис реалізованих інтерфейсів;
– результати виконання програми;
– аналіз взаємодії компонентів;
– висновки.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Основна

1. Phillips, Dusty. Python 3 Object-Oriented Programming: Build Robust and Maintainable Software with Object-Oriented Design Patterns in Python 3.8. 3rd ed. Packt Publishing, 2018. 466p. ISBN: 9781789617078, 1789617073.
2. Lott, Steven F. Mastering Object-Oriented Python. 2nd ed. Packt Publishing, 2019. 770p. ISBN: 9781789531404, 1789531403.
3. Lott, Steven F., Phillips, Dusty. Python Object-Oriented Programming. 4th ed. Packt Publishing, 2021. 714p. ISBN: 9781801075237, 1801075239.
4. Ramalho, L. Fluent Python. O'Reilly Media. 2022. 1014p. ISBN: 9781492056324, 1492056324.
5. Thomas, D., Hunt, A. The Pragmatic Programmer: Your Journey to Mastery, 20th Anniversary Edition. Pearson Education. 2019. 352. ISBN: 9780135956915, 0135956919.

Допоміжна

6. Martin, R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Pearson Education. 2018. 432p. ISBN 9780134494166.
7. Fowler, M. Refactoring: Improving the Design of Existing Code. Pearson Education. 2018. 448p. ISBN: 9780134757704, 013475770X.
8. Robert Johnson Object-Oriented Programming with Python: Best Practices and Patterns. 2024. HiTeX Press. 2024. 519p.
9. Sina, L. Algorithms and Data Structures. BoD - Books on Demand. 2025. 132p. ISBN: 9783819205330, 3819205330.
10. Goodrich, Michael T., Tamassia, Roberto, Goldwasser, Michael H. Data Structures and Algorithms in Python. Wiley, 2013. 768p. ISBN: 9781118290279, 1118290275.

Інформаційні ресурси

11. Python Software Foundation. Python Documentation. URL: https://docs.python.org
12. Python Enhancement Proposals (PEP). URL: https://peps.python.org
13. Real Python Tutorials. URL: https://realpython.com
14. Packt Programming Resources. URL: https://www.packtpub.com
15. Stack Overflow Developer Community. URL: https://stackoverflow.com

Навчальне видання

Педяш Володимир Віталійович

ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

Методичні вказівки до практичних робіт для здобувачів вищої освіти,
які навчаються за спеціальностями галузі «Інформаційні технології»

Українською мовою