**Практична робота №01. Порівняння мов програмування.**

**Посилання на всі лекції та лабораторні роботи**

**https://github.com/TetyanaLumpova/OP-AM/tree/main/2025**

**Мета**: навчитися писати аналізувати можливості мов програмування як інструменту розроблення власних програм**.**

**Завдання.**

Переглянути в Інтернет описи мов програмування С++, С#, Java. JavaScript, Pyton. Заповнити надану нижче таблицю для кожної мови, провести аналіз та надати свій висновок. В рядку "Основні особливості" вказати можливості мови, які відсутні в інших (конкретизувати, вказавши мову, яка не має таких властивостей). В рядку "Складність вивчення" визначте оцінку від 1 до 5, за якою ви визначаєте для себе складність вивчення цієї мови у порівнянні між цими мовами. У висновку визначити мову програмування, яку б ви хотіли використовувати, і причини, які спонукали вас зробити цей вибір.

|  |  |  |
| --- | --- | --- |
| Характеристика | Опис | Посилання на джерело |
| Область використання |  |  |
| Механізм компіляції |  |  |
| Парадигми, які використовуються |  |  |
| Основні особливості |  |  |
| Наявність стандарту мови |  |  |
| Складність вивчення |  |  |
|  |  |  |

Приклад оформлення посилань на сайти:

1. Конституція України (Відомості Верховної Ради України (ВВР), 1996, № 30, ст. 141) {Із змінами, внесеними згідно із Законами № 2222-IV від 08.12.2004, ВВР, 2005, № 2, ст. 44, № 2952-VI від 01.02.2011, ВВР, 2011, № 10, ст. 68, № 586-VII від 19.09.2013, ВВР, 12 2014, № 11, ст. 142, № 742-VII від 21.02.2014, ВВР, 2014, № 11, ст. 143, № 1401-VIII від 02.06.2016} // Вища рада правосуддя. Офіційний сайт. Нормативні акти. URL: http://www.vru.gov.ua/legislative\_acts/1 (дата звернення: 13.02.2019)
2. Мар’їна О. Контент-стратегія бібліотек у цифровому середовищі Бібліотечний вісник. 2016. № 4. С. 8–12. URL: http://nbuv.gov.ua/UJRN/bv\_2016\_4\_4 (дата звернення: 26.09.2017)
3. **Статистичний** щорічник за 2010 рік [Електронний ресурс] / Держ. служба статистики України. – Київ : Август Трейд, 2011. – Режим доступу : [http://library.oseu.edu.ua/docs/StatSchorichnyk Ukrainy 2010.pdf](http://library.oseu.edu.ua/docs/StatSchorichnyk%20Ukrainy%202010.pdf). – Назва з екрану (дата звернення: 21.10.2020)

Результати надсилати на електронну адресу викладача [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)у вигляді файлу з іменем у форматі

**<Номер групи><Номер лабораторної><Прізвище англійською>**

Наприклад, 21-01Ivanov.doc.

В темі листа вказати, номер групи, прізвище студента та номер ПР як "ЛР№1".

**Строк відсилки ЛР для ІПЗ-21 04.02.25**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

**ОП+АМ-Запитання-<Номер групи>-<Прізвище >**.

***Контрольні запитання*.**

1. Дайте визначення мови програмування.
2. Від чого залежить вибір мови програмування?
3. Що таке мнемокод?
4. Яка мова була першою широковживаною компільованою мовою? Яка ключова ідея лежала в її основі?
5. Яка з широковживаних мов програмування має найбільший вік?
6. Визначте покоління мов програмування.
7. На чому, на вашу думку, базується поділ мов програмування за поколіннями?
8. Зробіть класифікацію мов програмування
9. В чому полягає особливість імперативних мов програмування? Які мови відносяться до цієї групи?
10. В чому різниця між компілятором та інтерпретатором?
11. Наведіть приклади компіляторів та інтерпретаторів.

**Теоретична частина**

***Парадигми програмування***

Слово парадигма грецького походження і означає стиль міркування, спосіб дій або набір концепцій у певній галузі знань. Парадигми програмування – це моделі, які відтворюють спосіб мислення розробника програми. Мова програмування може підтримувати або не підтримувати ту чи іншу парадигму. В першому випадку застосування парадигми стає зручним, тобто простим, безпечним і ефективним, в іншому – складним і ненадійним.

***Паради́гма програмува́ння***— це підхід до побудови програм, в основі якого лежить використання концептуально узгодженої сукупності ідей і понять, що визначають певний стиль написання програми.

Кожна з парадигм програмування має свою концептуальну основу, використовує свій набір моделей і методів обчислень, включаючи структури даних і механізми управління, і з нею пов'язаний певний клас прикладних задач, які зручно вирішувати засобами даної парадигми. На тепер існує достатньо велика кількість парадигм програмування (рис. 1). Одні парадигми розвиваються незалежно, інші з'являються в результаті поєднання різних концептуальних ідей.

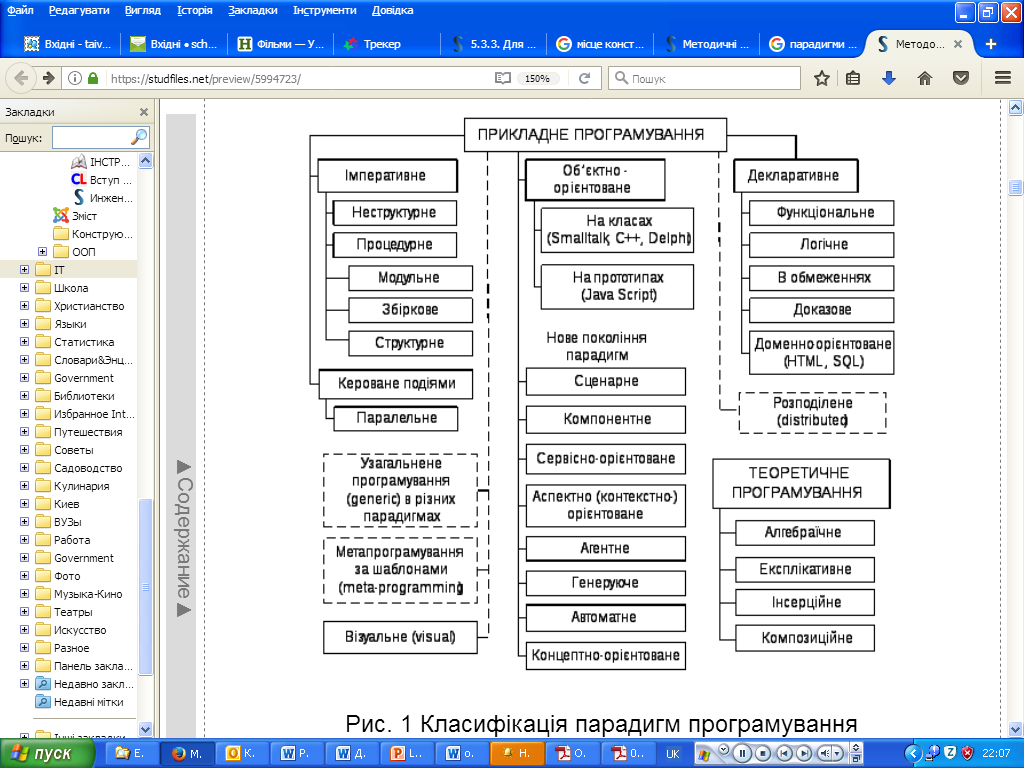


Рисунок 1 – Класифікація парадигм програмування

Основними серед існуючих парадигм програмування вважаються наступні:

* Імперативне програмування;
* об'єктно-орієнтоване програмування;
* декларативне програмування.

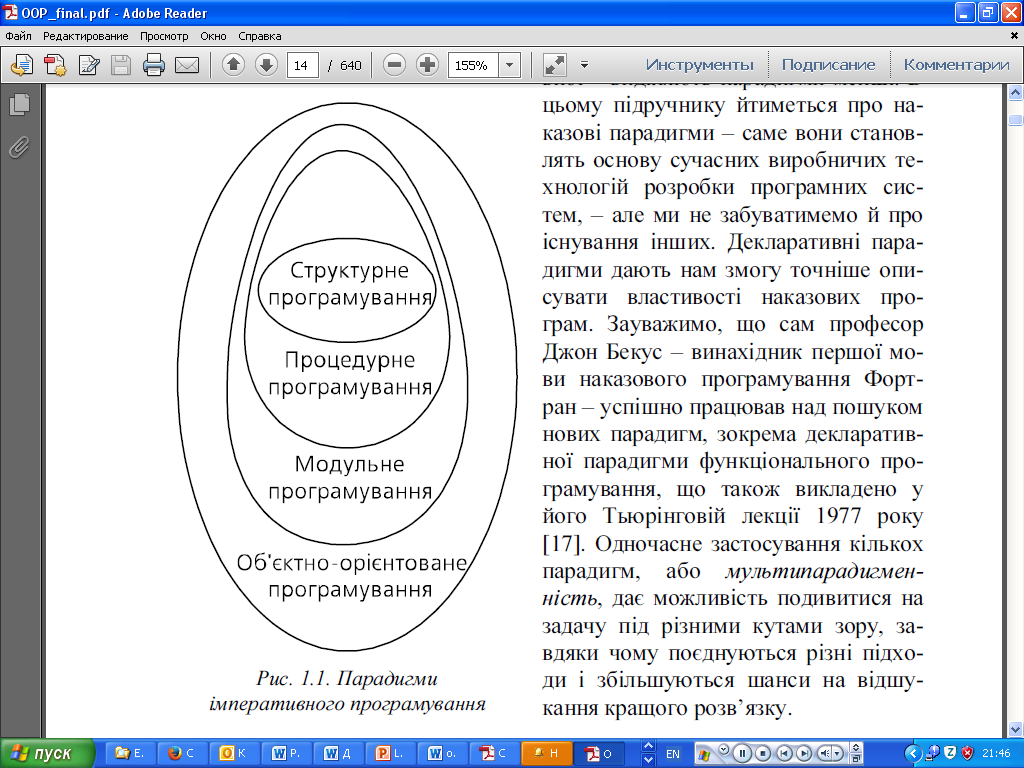


Рисунок 2 – Парадигми імперативного програмування

**Імперативне програмування –** це підхід до побудови програм, що використовує алгоритмічну декомпозицію задачі, при якій програма представляється як послідовність дій (команд), які повинен виконати комп'ютер покроковим чином.

Концептуальною ідеєю імперативного підходу до розробки програм є *алгоритмічна* структура програмного коду*, яка передбачає роз'єднання у програмах даних і дій, що виконуються над ними.* Активним суб’єктом у імперативних програмах вважається алгоритм, який повинен виконати всі необхідні для досягнення потрібного результату дії над пасивними даними. Таким чином, наголос у імперативному програмуванні робиться на верховенстві алгоритму над даними.

Імперативна парадигма програмування є відображенням архітектури традиційних ЕОМ, яка була запропонована фон Нейманом. Теоретичною основою імперативного програмування служить алгоритмічна система під назвою «машина Т’юринга» - абстрактний обчислювальний пристрій, що виконує послідовність команд програми, яка, таким чином, переходить з одного стану в інший. Відповідно програма в імперативному програмуванні розглядається як послідовність дій, які змінюють стан комп‘ютера.

Якщо розглядати стан комп'ютера як стан пам'яті, тобто значень комірок пам'яті, то імперативна програма – це послідовність операторів, які перетворюють вихідний стан пам'яті, тобто значення вихідних даних, у заключний, тобто у результати. Концепції пам'яті як сховища значень, поточного кроку виконання і поточного стану, що змінюється у часі, є фундаментальними в імперативному програмуванні.

Моделлю обробки даних при імперативному підході є послідовне виконання команд, які задають алгоритм вирішення задачі. Коротко це можна представити наступною нотацією: **програма = послідовність дій**.

Базовими поняттями імперативної парадигми програмування є поняття *оператора* (команди), що задає дію по обробці даних, та поняття *змінної*, якій може бути присвоєно значення, що зберігається в пам‘яті комп'ютера; базовою операцією є операція присвоєння, що служить для зміни вмісту областей пам'яті.

Основна задача програміста при написанні імперативної програми – звести рішення задачі до послідовності операторів, які вміє виконувати процесор комп'ютера.

Кожна парадигма програмування має відповідну інструментальну підтримку. Найбільш відомі і поширені імперативні мови програмування Фортран(1954), Алгол (1960), Паскаль(1970), С(1972).

Оскільки практично всі сучасні комп'ютери орієнтовані на послідовні обчислення, імперативне програмування явно виграє в ефективності реалізації прикладних задач, для яких важлива швидкість виконання. Окрім цього, робота із зовнішніми пристроями, як правило, описується в термінах послідовного виконання операцій, що робить такі задачі ідеальними для імперативної реалізації.

**Об'єктно-орієнтоване програмування** – це підхід до побудови програм, що використовує об'єктну декомпозицію задачі, при якій структура системи описується в термінах об'єктів і зв'язків між ними, а поведінка - в термінах обміну повідомленнями між об'єктами.

Об'єкт представляє собою цілісну модель і природну імітацію діяльності деякого елемента реального світу. Об'єкти, як і реальні сутності навколишнього світу характеризуються тим, що вони мають певний набір властивостей, здатні різними способами змінювати ці властивості і можуть реагувати на події, які виникають як у навколишньому світі, так і усередині самого об'єкта. Об'єкт, як правило, включає деякі дані, які характеризують стан об'єкта, та функції обробки цих даних, які описують поведінку об'єкта. Функції об'єкта, які у об'єктно-орієнтованому програмуванні (ООП) називають *методами*, зазвичай призначені для доступу до даних об'єкта та виконання певних дій над ними.

Поєднання даних і дій, що виконуються над ними, в єдине ціле, яке називають *об'єктом*, і є концептуальною ідеєю ООП.

Кожен об'єкт є екземп*ляром* певного *класу.* Клас в ООП - це в чистому вигляді *абстрактний тип даних*, що створюється програмістом. Клас представляє собою множину об'єктів зі схожою структурою і схожою поведінкою. *Усі об'єкти, які є екземплярами одного класу, можуть виконувати одні й ті ж самі дії.* Поняття *об'єкта* та *класу* є фундаментальними в ООП.

Об’єкти взаємодіють між собою шляхом надсилання *повідомлень*. Об'єкт, що приймає повідомлення, реагує на нього особливим, тільки йому відомим чином. При цьому він може послати повідомлення іншим об'єктам, отримати від них відповіді, змінити свій стан і, нарешті, повернути відповідь тому об'єкту, який послав йому повідомлення. Всі дії та розрахунки в об'єктно-орієнтованих програмах виконуються шляхом взаємодії (обміну даними) між об'єктами за допомогою механізму передачі повідомлень (рис. 3,4).

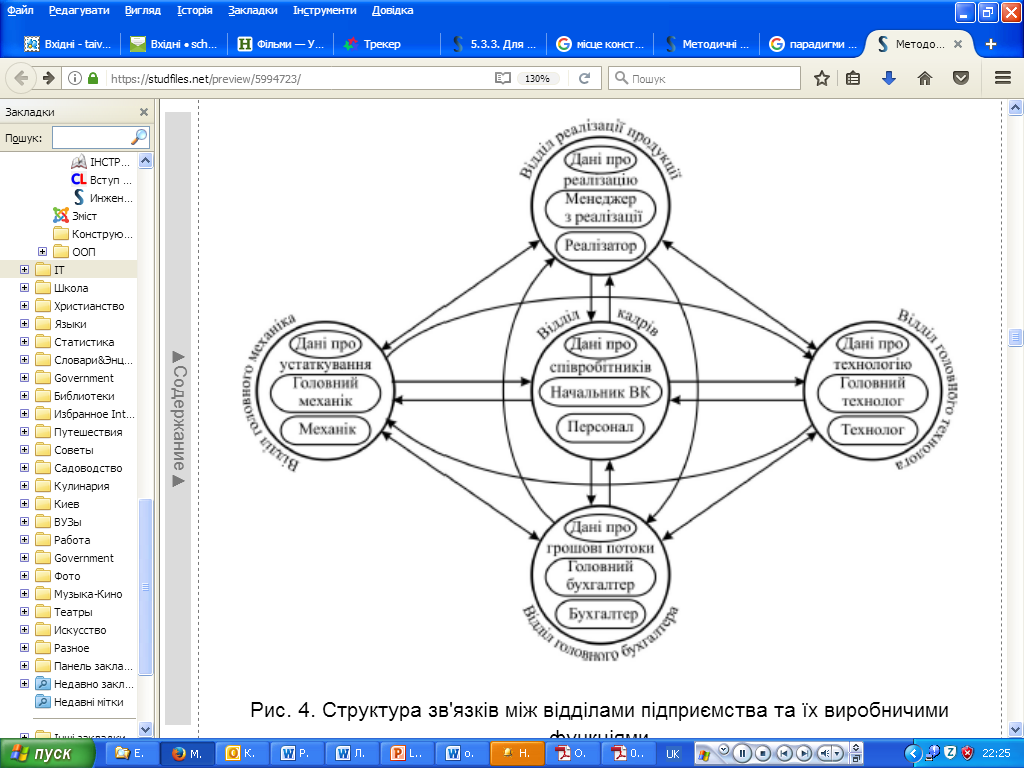
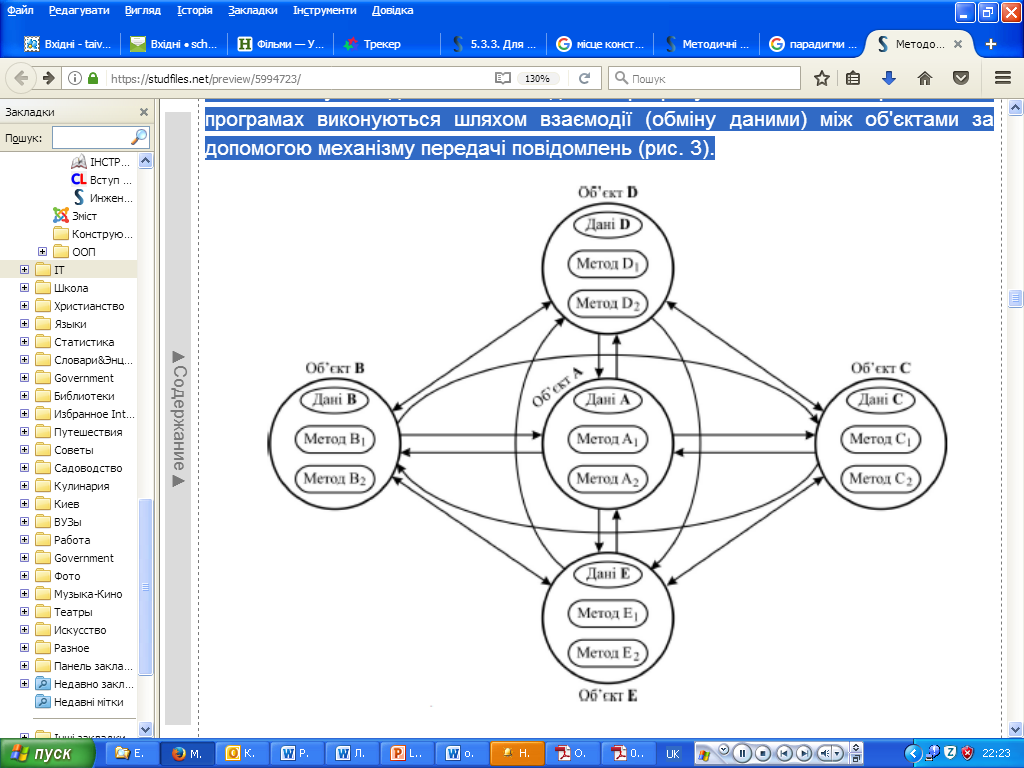


Рисунок 3 – Опис класів Рисунок 4 – Структура зв‘язків між

підрозділами підприємства на основі опису

класів

ООП ніяк не пов'язане з процесом виконання програми, а є тільки способом її ефективної організації. Основою такої організації є використання принципів *інкапсуляції*, *успадкування* та *поліморфізму*.

***Інкапсуляція*** визначає спосіб опису, який передбачає об'єднання в межах класу даних і методів їхньої обробки, а також приховування деталей реалізації класу: користувач повинен бачити і використовувати для доступу до об'єкту і маніпулювання ним виключно інтерфейсну частину класу (список декларованих властивостей і методів). Це дозволяє забезпечити захист даних від зовнішнього втручання.

***Успадкування*** - це механізм породження нових класів з уже існуючих. Породжений клас успадковує властивості та поведінку класу-предку і може доповнити їх своїми власними. Механізм успадкування дозволяє *організовувати класи у єдину деревоподібну структуру із загальним коренем, яка називається ієрархією успадкування і характеризуються тим, що в*ластивості *та поведінка, зв'язані з екземплярами деякого класу, автоматично доступні будь-якому класу, розташованому нижче в ієрархічному дереві. Це дає змогу* будувати похідні поняття на основі базових і таким чином створювати модель як завгодно складної предметної області із заданими властивостями. Наприклад, ієрархія класів - геометричних фігур:

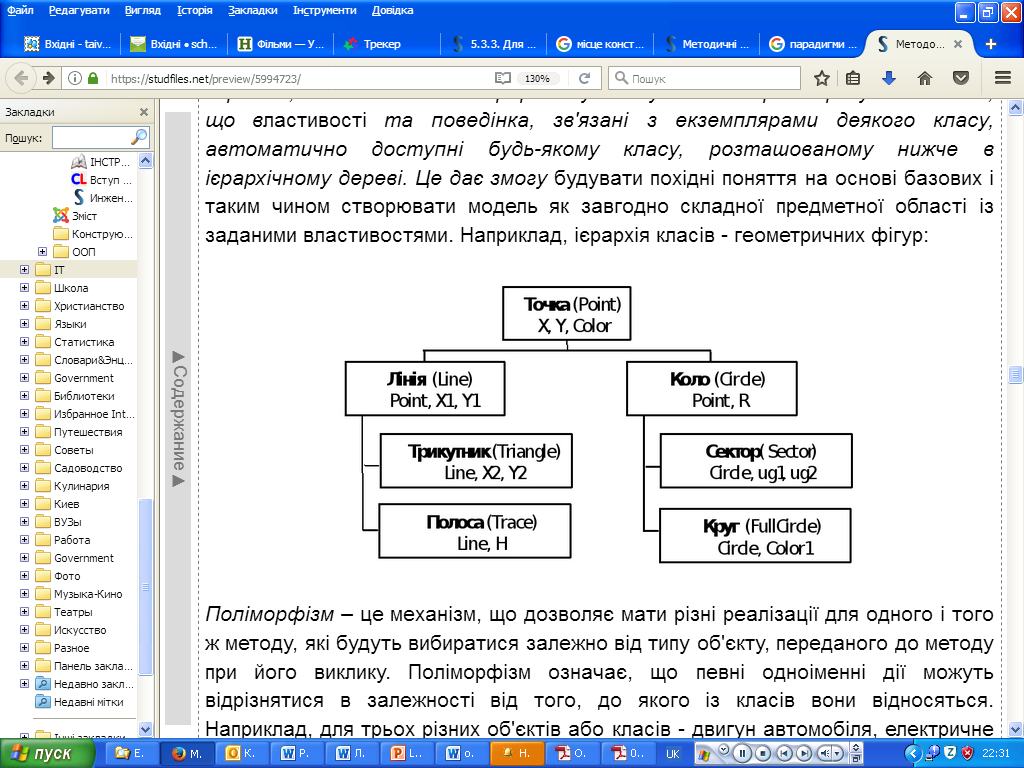


Рисунок 5 – Приклад реалізації механізму успадкування

***Поліморфізм*** – це механізм, що дозволяє мати різні реалізації для одного і того ж методу, які будуть вибиратися залежно від типу об'єкту, переданого до методу при його виклику. Поліморфізм означає, що певні однойменні дії можуть відрізнятися в залежності від того, до якого із класів вони відносяться. Наприклад, для трьох різних об'єктів або класів - двигун автомобіля, електричне світло в кімнаті і комп'ютер, можна визначити операцію "вимкнути". Проте сутність цієї операції буде відрізнятися для кожного із розглянутих об'єктів. Використання поліморфізму робить програмне забезпечення більш гнучким і універсальним.

Таким чином, на відміну від імперативного програмування, що базується на алгоритмічній декомпозиції програми, а також роз'єднанні у програмах даних і процедур їхньої обробки, у об'єктно́-орієнтованих програмах має місце об'єктна декомпозиція програми і об’єднання даних та функцій їхньої обробки в межах певного класу. Первинними в ООП є дані (об'єкти), а не код. Об'є́ктно-орієнто́вана програма фактично є описом структури і поведінки окремих об'єктів, які взаємодіють між собою таким чином, щоб забезпечити певну поведінку системи.

Стійкість та керованість об'єктно-орієнтованої програми забезпечуються за рахунок чіткого розподілу відповідальності об’єктів (за кожну дію відповідає певний об’єкт), однозначного означення інтерфейсів міжоб’єктної взаємодії та повної ізольованості внутрішньої структури об’єкта від зовнішнього середовища (інкапсуляції).

Обчислювальна модель чистого об'єктно-орієнтованого програмування підтримує явно тільки одну операцію, якою є посилка об'єкту повідомлення. Повідомлення можуть мати параметри, які є об'єктами. Саме повідомлення також є об'єктом.

Роль програміста при написанні об'єктно́-орієнтованої програми полягає у формуванні і реалізації такої ієрархії об'єктів, взаємодія яких після запуску програми приведе до досягнення необхідного кінцевого результату. Використання раніше розроблених (можливо, іншими колективами програмістів) бібліотек об'єктів дозволяє значно заощадити трудовитрати при розробці програмного забезпечення, особливо типового.

Об'єктно-орієнтовані мови можна розділити на три групи

* *Чисті мови*, які в найбільш класичному вигляді підтримують тільки одну об'єктно - орієнтовану парадигму програмування. Такі мови містять невелику мовну частину і істотну бібліотеку, а також набір засобів підтримки часу виконання. Приклади: Simula (1962), Smalltalk (1972)
* *Гібридні мови*, які з'явилися в результаті впровадження об'єктно-орієнтованих конструкцій в популярну імперативну мову програмування C++(1983), ObjectPascal(1984)
* *Урізані мови,* які з'явилися в результаті видалення з гібридних мов найбільш небезпечних і непотрібних з об'єктно-орієнтованої точки зору конструкцій С#(2000), Java(1995)

**Декларативне програмування**—це парадигма програмування, відповідно до якої програма деяким чином описує ***що*** потрібно отримати як результат, а не ***як*** це треба зробити. Тому декларативне програмування часто називають *описовим*. Тут головним є чітке формулювання мети і результатів роботи задачі, а не послідовність отримання цього результату. Вибір і застосування необхідного для вирішення задачі алгоритму - проблема виконуючої системи (алгоритм роботи з даними "зашитий" в неї).

Так, декларативним є опис web-сторінок на HTML — вони описують, що містить сторінка та *що* має відображатись (заголовок, шрифт, текст, зображення), але не містять інструкцій *як* її слід відображати; опис SQL-запитів, які конкретизують властивості даних, які слід отримати від бази даних, але не процес отримання цих даних; опис XML-документів тощо.

Такий підхід має високий ступінь абстракції і легко формалізується математичними засобами. Фактично програміст оперує не набором інструкцій, а абстрактними поняттями, часто досить узагальненими.

Декларативні мови найкраще використовувати у випадках, коли "дані управляють програмою": при написанні експертних систем, при конструюванні трансляторів з мов програмування, для більшості задач штучного інтелекту. Саме там їх використання призводить до найбільшої ефективності.

Узагальнимо основні властивості розглянутих парадигм програмування:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Парадигма* | *Ключовий концепт* | *Програма* | *Виконання програми* | *Результат* |
| **імперативна** | оператор | послідовність операторів | виконання операторів | результуючий стан пам’яті |
| **об’єктно-орієнтована** | **об’єкт** | набір класів о**б’єктів** | обмін повідомленнями між о**б’єкт**ами | результуючий стан о**б’єктів** |
| **функціональна** | функція | набір функцій | обчислення функцій | значення головної функції |
| **логічна** | предикат | логічні формули | логічний доказ | результат доказу |

Парадигма програмування не визначається однозначно мовою програмування — багато сучасних мов програмування є *мультипарадигменними*, тобто допускають використання декількох різних парадигм програмування. Зокрема, мова C++ є універсальною мовою програмування високого рівня з підтримкою декількох сучасних парадигм програмування: процедурної, об'єктно-орієнтованої та узагальненої.

Розвиток парадигм пов'язаний з двома основними взаємозв'язаними причинами:

1. розширенням класу функціональних задач і пошуком найбільш ефективних методів їх вирішення;
2. зростанням складності програм і систем, зростанням вимог до їхньої якості та надійності.

Мистецтво програмування якраз і полягає в тому, щоб вибрати одну з мов, яка найкраще підходить для вирішення наявної задачі.

***Методологія програмування***- це об'єднана єдиним філософським підходом сукупність методів, що застосовуються в процесі створення програм.

З кожною методологією можна пов'язати деякі характерні для неї атрибути, а саме:

* парадигму програмування, що визначає основне джерело ефективності методології;
* узгоджену, зв'язну множину методів, через які реалізується дана методологія;
* концепції (поняття, ідеї), що підтримують методи і дозволяють більш точно їх визначити.

***Метод***–це сукупність теоретичних принципів і практичних прийомів, які потрібно застосувати для виконання певної задачі.

Так основними методами, через які реалізується імперативна парадигма програмування, є:

* *метод зміни станів*, що полягає в послідовному зміні станів пам’яті (змінних) і підтримується концепцією алгоритму;
* *метод управління потоком виконання*, який полягає в покроковому контролі управління і підтримується концепцією потоку виконання.

Самий ранній стиль розробки імперативних програм характеризувався *неструктурованістю* - код програми представлявся єдиним безперервним блоком; переходи до потрібних секцій програми виконувалися за допомогою операторів переходу (зазвичай, goto). Подібні проекти отримали назву BS-програм – абревіатура від "bowl of spaghetti" – блюдо спагеті, бо саме так виглядала готова програма при спробі зобразити всі переходи між її операторами.

Хорошими програмістами на той час вважалися ті, хто створював достатньо хитромудрі програми, які займали мінімальний об'єм пам'яті та швидко виконувалися. Це було цілком природно, враховуючи тодішні можливості обчислювальної техніки. Однак, такі програми було важко (якщо взагалі можливо) зрозуміти іншим фахівцям. Інколи навіть автори таких програм з плином часу з трудом розуміли власне творіння. Тому при розробці таких програм виникали проблеми забезпечення якості: утворення «спагетті-коду», складність його перевірки і тестування.

Із збільшенням складності задач розмір програм постійно збільшувався. Типовою стала ситуація, коли в різних місцях програми виникала необхідність у виконанні однієї і тієї ж послідовності дій. Такі дії могли бути досить складними і представлятися великими фрагментами програми. Для забезпечення компактності і підвищення наочності програм було запропоновано фрагменти програми, що повторюються, оформляти у вигляді спеціальних синтаксичних конструкцій – *підпрограм*(процедур або функцій), які за необхідності могли бути викликані для виконання з будь-якого місця основної програми. Методологія програмування, заснована на концепції виклику підпрограми, що є логічно завершеним фрагментом програми, отримала назву ***процедурного програмування***.

Основними ідеями, на яких грунтується процедурне програмування, є:

* використання механізмів передачі параметрів і повертання результатів;
* розподіл областей дії змінних програми (локальні, глобальні),
* використання різних моделей пам'яті для зберігання змінних;
* блокова організація програм.

Основними методами, через які реалізується процедурне програмування, є:

* *метод процедурної декомпозиції*, який полягає у оформленні кожного логічно завершеного фрагменти програми у вигляді підпрограми, і грунтується на концепції виклику підпрограми;
* *метод блочності*, що розглядає програму як деяку ієрархічну структуру, яка складається з головної програми і множини вкладених блоків (підпрограм), і грунтується на концепції блокової організації програм;
* *метод локалізації даних*, який полягає у видимості (доступності) даних, що обробляються у підпрограмі, лише в межах підпрограми, і грунтується на концепції розподілу областей дії змінних програми;
* *метод специфікації інтерфейсів*, який полягає у формалізованому описі входів, виходів і підпрограм;
* *механізм раннього зв’язування*, який полягає у статичному зв’язуванні підпрограм (на етапі компіляції).

Моделлю обробки даних, характерною для процедурного програмування, є послідовне виконання підпрограм. Коротко це можна представити наступною нотацією:

***програма = послідовність процедур, кожна з яких є послідовністю елементарних дій і викликів процедур***.

Використання процедурного підходу при розробці програм дозволяє зменшити об'єм програмного коду, виключивши його дублювання, зробити програми прозорішими, прискорити процес їх написання і тестування. Використання підпрограм дозволило розбивати великі задачі на підзадачі і таким чином спростило написання великих програм. Завдяки даному підходу програми набули більш впорядкованого характеру.

Наступним кроком розвитку програмування стало підвищення структурованості програм – ***структурне програмування***. Дана методологія - це сукупність методів проектування та написання програм за жорсткими правилами, дотримання яких підвищує продуктивність праці програмістів, поліпшує читабельність і полегшує процес тестування програм.

Витоки цієї методології сягають70-х років XX століття, коли швидкими темпами почала зростати складність програм, а отже, постала потреба у методах подолання такої складності. Вона ґрунтується на теоремі Бома-Якопіні (1966 р.), які довели, що будь-який виконуваний алгоритм може бути перетворений до структурованого вигляду, тобто такого виду, коли хід його виконання визначається тільки за допомогою трьох структур управління: послідовностей, розгалужень і циклів.

Методологія структурного програмування реалізується через методи:

* структурного кодування (структурування програм);
* низхідного проектування (спадного проектування, проектування «зверху-вниз»);
* блочного програмування;
* паралельного документування.

*Метод структурного кодування* передбачає, що будь-яка програма представляє собою структуру, побудовану із трьох типів базових конструкцій:

* *послідовне виконання*– одноразове виконання операцій в тому порядку, в якому вони записані в тексті програми;
* *розгалуження*– одноразове виконання однієї із двох або більшої кількості операцій, в залежності від виконання деякої заданої умови;
* *цикл*– багаторазове виконання однієї і тієї ж операції до тих пір, доки виконується деяка задана умова (умова продовження циклу).

В програмі базові конструкції можуть бути вкладені одна в одну довільним чином, але ніяких інших засобів керування послідовністю виконання операцій не передбачається. Зокрема, неприпустимим є використання оператора безумовного переходу (GOTO).

Метою структурування є перетворення неструктурованої програми на еквівалентну їй структуровану, тобто таку, що складається з обмеженого набору керуючих алгоритмічних структур. Методи структурування ґрунтуються на поняттях функціонального вузла, а також на поняттях простої, елементарної і складеної програми.

Елементарна програма — це проста програма, що не містить блоків, які складаються більше ніж з одного вузла. Слід зазначити, що існує обмежена кількість елементарних програм, з яких основними є три програми, які зображені на рис. 6.

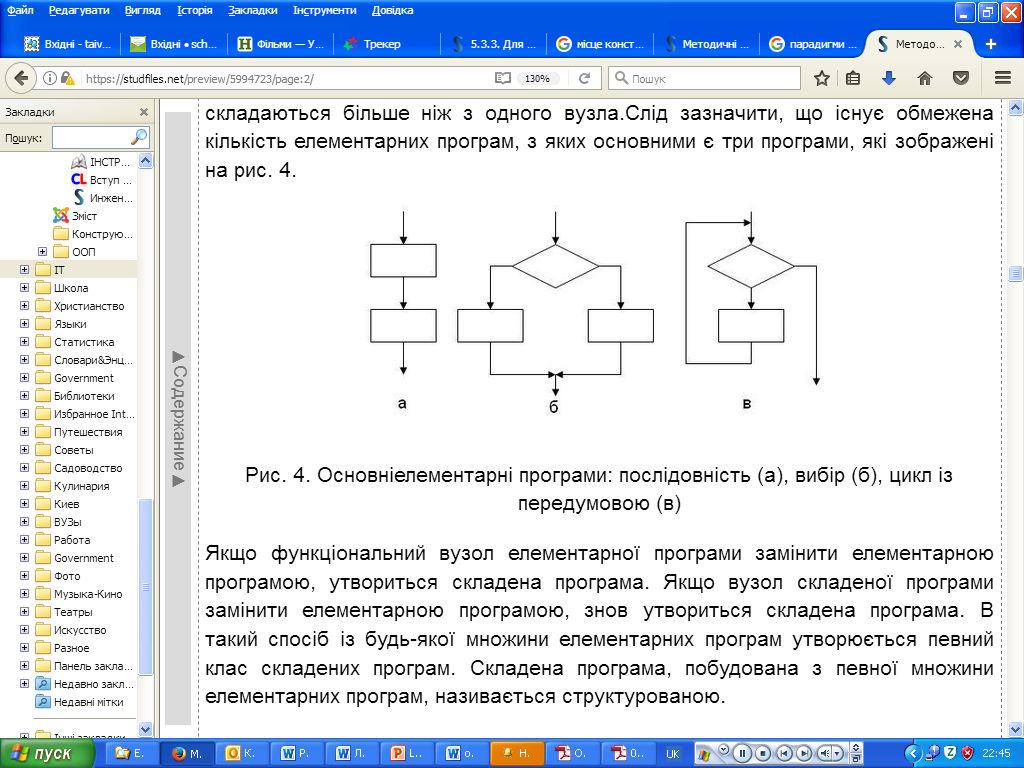


Рисунок 6 – Основні елементарні програмні структури: а) послідовне виконання (слідування); б) розгалуження (вибір); в) цикл (з передумовою)

Якщо функціональний вузол елементарної програми замінити елементарною програмою, утвориться складена програма. Якщо вузол складеної програми замінити елементарною програмою, знов утвориться складена програма. В такий спосіб із будь-якої множини елементарних програм утворюється певний клас складених програм. Складена програма, побудована з певної множини елементарних програм, називається структурованою.

Позначимо літерою S клас складених програм, базисна множина якого містить такі елементарні програми, як послідовність (рис. 6, а), вибір (рис. 6, б) і цикл із передумовою ( рис. 6, в). Теорема про структурування стверджує, що будь-яку просту програму шляхом покрокового перетворення можна замінити функціонально еквівалентною структурованою програмою, що належить означеному вище класу S. Згадане покрокове перетворення здійснюється за переліченими нижче правилами:

1. *правило простоти* - створення програми слід починати із простої програми;
2. *правило пакетування* -кожну дію можна замінити двома послідовними діями (вихід одного функціонального вузла з'єднується із входом наступного);
3. *правило вкладання*–кожну дію можна замінити будь-якою структурою керування (будь-який блок може бути замінений на структуру керування вибору або повторення);
4. правила 2 і 3 можна застосовувати у будь-якій послідовності необмежену кількість разів.

Принцип застосування правил пакетування та вкладання до простої програми продемонстровано на рис. 7.

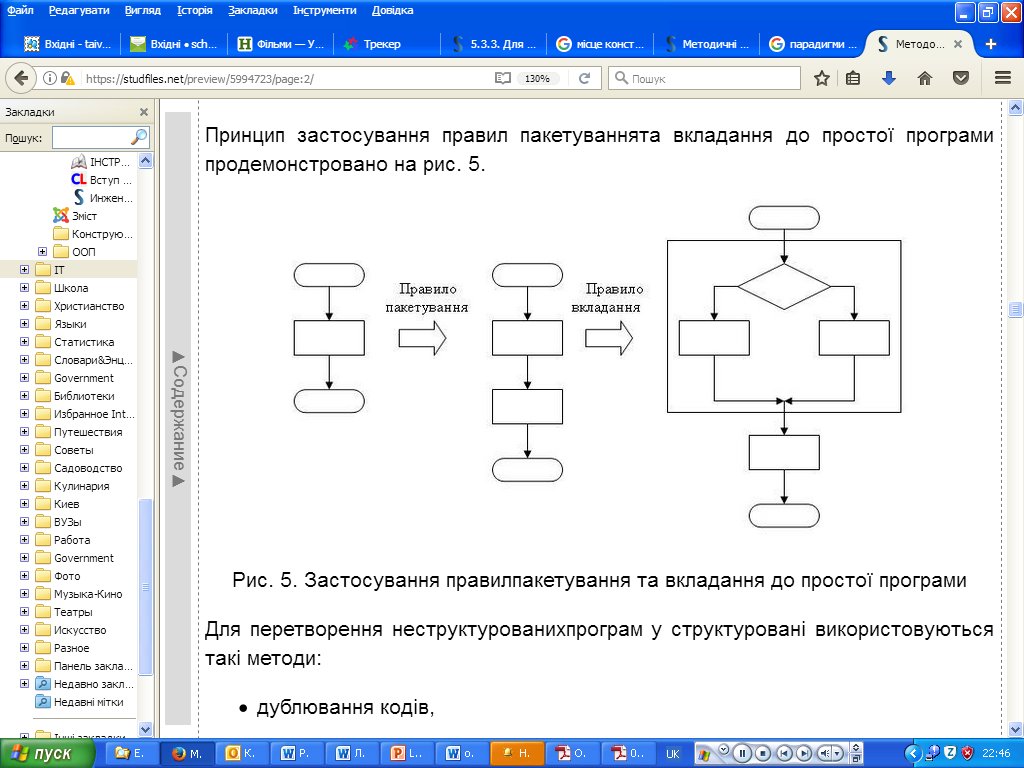


Рисунок 7 – Застосування правил пакетування та вкладення до простої програми

Для перетворення неструктурованих програм у структуровані використовуються такі методи:

* дублювання кодів,
* введення змінної стану,
* метод булевих ознак.

*Метод дублювання кодів* застосовується для перетворення неструктурованих програм, що не містять циклів. Для отримання структурованої програми дублюють ті блоки (модулі), у які можна увійти з декількох точок. (Це не найкраща практика)

*Метод булевої ознаки* використовується для перетворення неструктурованих програм, що містять цикли. В програму вводиться додаткова змінна, яка набуває значення true чи false. Доки змінна ознаки зберігає це значення, виконання циклу триває. Значення змінної ознаки всередині циклу модифікується лише за певних умов.

*Блочне програмування* — це організація програми у вигляді сукупності відносно незалежних складових частин - блоків, якими фактично є підпрограми. Як показує практика, великі задачі простіше вирішуються, якщо розглядати їх як сукупність менших задач.

Як правило, програмний блок розв’язує порівняно нескладну задачу, логічно незалежну від інших задач. Його властивості:

* відповідає лише одній задачі;
* має один вхід і один вихід;
* має порівняно невеликий розмір;
* доступний за своїм ідентифікатором;
* може викликати інший блок(не в усіх мовах програмування);
* підтримує незалежність функціонування (заміна підпрограми на аналогічну не впливає на всю програму).

Тобто, кожна програма представляється лінійною послідовністю блоків. Кожен блок складається з одного або декількох інших блоків, кожен із яких має також рівно один вхід і рівно один вихід і сам може розглядатися як блок. Сенс блоків, що мають один вхід і один вихід, полягає в тому, що можна вийняти їх, змінити їх начинку і вставити назад без зміни інших сполучень в програмі.

Блоки повинні бути незалежними в межах інтерфейсу програми і структури даних. Практика показала, що чим вищий ступінь незалежності підпрограм, тим простіше розібратись в окремих програмних блоках і в програмі в цілому; тим менша ймовірність так званого *хвильового ефекту*– появи нових помилок при виправленні старих або при внесенні змін у програму. Тому не варто без крайньої необхідності використовувати в підпрограмах глобальні змінні. Всі зв'язки між підпрограмами мають підтримуватися через списки параметрів.

В основу *методу низхідного проектування* покладено структурну декомпозицію7задачі із застосуванням принципу покрокової (поетапної) деталізації. Спадне проектування передбачає, що розробка програм ведеться методом «зверху вниз», від загального до деталей. Спочатку задача розглядається як єдиний блок, що виражає загальне призначення програми. Далі ця задача поділяється на декілька дрібніших підзадач в тому порядку, в якому вони повинні виконуватися. Потім кожна з підзадач розбивається на свої підзадачі, що належать другому рівню деталізації і т.д. Процес покрокової деталізації підзадач здійснюється до тих пір, поки підзадачі чергового рівня не стануть досить простими для незалежного розв’язання (наприклад, кожній із них буде відповідати окрема команда мови програмування).

*Процедурне структурне програмування* передбачає використання принципу *процедурної декомпозиції* на всіх рівнях проектування програмної системи. У відповідності із даним принципом кожен логічно цілісний програмний блок, отриманий на певному рівні деталізації, оформлюється як *підпрограма*. Наприклад, підпрограма обчислення визначника матриці, підпрограма знаходження суми елементів ряду тощо.

Розробка процедурної структурованої програми ведеться покроково, методом "зверху вниз" з використанням механізму так званих «заглушок» - підпрограм, які нічого не роблять. При цьому спочатку пишеться основна програма, у якій замість кожного логічного зв'язного фрагмента тексту вставляється виклик підпрограми, яка буде виконувати цей фрагмент. Замість справжніх, працюючих підпрограм, в програму вставляються "заглушки". Отримана програма перевіряється та налагоджується. Після того, як програміст переконається, що підпрограми викликаються в правильній послідовності, тобто загальна структура програми вірна, підпрограми-"заглушки" послідовно замінюються на реально працюючі, причому розробка кожної підпрограми ведеться тим же методом, що і основної програми. Розробка закінчується тоді, коли не залишиться жодної "заглушки", яка не була б видалена.

Таким чином, «заглушки» дозволяють перевірити логіку програми верхнього рівня до реалізації наступного. Тобто на кожному кроці розробки програми існує працездатний «каркас», який поступово обростає деталями. Така послідовність гарантує, що на кожному етапі розробки програміст одночасно має справу з доступною для огляду і зрозумілою йому множиною фрагментів і може бути впевнений, що загальна структура всіх більш високих рівнів програми вірна. При супроводженні та внесенні змін у програму з'ясовується, в які саме процедури потрібно внести зміни, і вони вносяться, не зачіпаючи безпосередньо не пов'язані з ними частини програми. Це дозволяє гарантувати, що при внесенні змін і виправленні помилок не вийде з ладу якась частина програми, що знаходиться в даний момент поза зоною уваги програміста.

Таким чином, при структурному програмуванні програма ієрархічно структурується і розробляється шляхом послідовного уточнення на кожному рівні ієрархії. В основу цього процесу, окрім принципу ієрархічності, покладено принципи абстрагування.

*Абстрагування*— це спрощений опис системи, в якому зосереджують увагу на певних властивостях і деталях, а на інші не зважають. Вдалою є та абстракція, що підкреслює суттєві деталі і відкидає несуттєві. Під час низхідного проектування програми на верхніх рівнях абстракції деталі реалізації приховуються, а на нижніх рівнях вони описуються конкретною мовою програмування.

Фактично такий підхід збільшив структурованість програм – велика програма стала сукупністю процедур-підпрограм. Одна підпрограма, головна, розпочинала роботу всієї програми.

Методи низхідного проектування та блочного програмування регламентують процес побудови архітектури програм на макрорівні. Методи структурування, навпаки, працюють на мікрорівні, регламентуючи процес створення програмного коду.

Сутність методу *паралельного документування* полягає у тому, що при розробці програм документація повинна створюватися одночасно із програмуванням, зокрема, у вигляді коментарів до програми. Документування програмного тексту дозволяє будь-якому програмісту легко читати і розуміти його у процесі розробки та подальшого супроводу.

Впровадження принципів структурного програмування зробило тексти програм, навіть досить великих, більш читабельними. Серйозно полегшилось розуміння програм, з'явилася можливість розробки програм в нормальному промисловому режимі, коли програму може без особливих труднощів зрозуміти не тільки її автор, а й інші програмісти. Все це полегшило і прискорило процес розробки програмного забезпечення.

Розвитком ідей процедурного програмування, пов'язаним із зростанням об'єму і складності програмного забезпечення, стало так зване ***модульне програмування***. Основна ідея модульного програмування полягає у організації програм як сукупності незалежних блоків, які прийнято іменувати *модулями*.

Визначальним принципом модульного програмування є незалежність модулів. Це означає, що програма має бути поділена на модулі таким чином, щоб зв'язки між модулями були слабкими, а зв'язки всередині модулів — сильними. Модулі вважаються незалежними, якщо вони задовольняють таким вимогам:

* кожен модуль можна замінити іншим функціонально еквівалентним модулем, що має той самий інтерфейс;
* взаємозв'язки між модулями встановлені за ієрархічним принципом;
* усі структури даних інкапсульовані в модулі, тобто доступ до даних може здійснюватися лише через процедури та функції модуля;
* модуль має одну точку входу та одну точку виходу;
* модуль повертає керування тому програмному блоку, що його викликав.

При розбитті програмного забезпечення на модулі для кожного модуля визначається функціональність, що реалізовується ним, а також зв'язки з іншими модулями. При цьому функціональність модуля (**що** він повинен виконувати) відокремлюється від його реалізації (**як** це виконується). Зв'язки між модулями здійснюються через спеціальний інтерфейс, тоді як доступ до реалізації модуля (тіл підпрограм і деяких «внутрішніх» змінних) заборонений. Тобто, знаючи, що робить модуль і як його викликати, можна без обмежень використовувати його, не маючи при цьому жодного уявлення, як саме він реалізований. Такий підхід розмежовує доступ до даних програми, зменшуючи кількість помилок, що виникають при роботі з ними.

Ще одна ідея модульного програмування полягає угрупуванні підпрограмі даних, якими вони управляють, в окремо компільовані файли, які також називають модулями. Основна програма може підключати такі модулі і використовувати підпрограми, що містяться в них. Тим самим забезпечується можливість автономної реалізації і зберігання коду (в бібліотеках модулів), а також збирання програм з модулів.

Основними ідеями, на яких грунтується модульне програмування, є:

* автономна реалізація модулів (в бібліотеках модулів),
* автономна компіляція,
* збирання програм з модулів.

Основними методами, через які реалізується модульне програмування, є:

* *метод незалежності модулів*, який полягає у розробці модулів, відносно ізольованих один від одного;
* *механізм пізнього зв’язування*, який полягає у динамічному виклику модулів (на етапі виконання).

Розбиття на модулі зменшує час перекомпіляції і полегшує процес налагодження програм, приховуючи несуттєві деталі за інтерфейсом модуля і дозволяючи налагоджувати програму по частинах. Інтерфейсом модуля є заголовки всіх підпрограм і описи доступних ззовні типів, змінних і констант. Використання модульного програмування істотно спрощує розробку програмного забезпечення, зокрема, шляхом розподілу процесу його розробки між групами розробників. Крім того, модулі надалі без змін можна використовувати в інших розробках, що підвищує продуктивність роботи програмістів. У разі колективної розробки модульної програми робота розподіляється так: більш досвідчені програмісти відповідають за інтерфейс модулів, а менш досвідчені — за їх реалізацію.

Модульний принцип покладено в основу практично всіх стилів програмування.