УДК 004.432

**О. І. Черняк**

# АСПЕКТИ КРИТИЧНОГО ПІДХОДУ ДО ВИКЛАДАННЯ ПОНЯТТЯ ПОЛІМОРФІЗМУ В ОБ'ЄКТНО-ОРІЄНТОВАНОМУ ПРОГРАМУВАННІ

Вінницький національний технічний університет

***Анотація***

*Запропоновано аспекти подання інформації про поліморфізм в ООП на основі критичного підходу який полягає у визначенні причин вибору даної парадигми, її переваг і недоліків, аналізу проблем на цьому напрямку, розкриття механізмів поліморфізму та аналізу їх ефективності.*

**Ключові слова**: об'єктно-орієнтоване програмування, методологія викладання, поліморфізм.

***Abstract***

*The aspects of polymorphism teaching in OOP based on critical approach which consist in defining reasons of choosing it paradigm, his advantages and lacks, analyzing problems in this way, explaining mechanisms of polymorphism and analyzing their efficiency are suggested.*

**Keywords**: object-oriented programming, teaching methodology, polymorphism.

## Вступ

Використання об'єктно-орієнтованого підходу при побудові програмного забезпечення призвело до значного підвищення ефективності програмування, що в період бурного розширення сфер використання обчислювальної техніки стало вирішальним чинником успішного впровадження програмних засобів у виробничій діяльності і побуті людини. З'явившись вперше у мові C++, даний підхід забезпечив такі переваги на етапі розробки програм, що отримав назву об'єктно-орієнтованого програмування (ООП) і по суті став еталоном при розробці складних програмних комплексів [1]. На даний момент практично кожна сучасна мова програмування високого рівня в тому чи іншому виді реалізує принципи ООП. Тому глибоке розуміння студентами ООП і знання техніки його використання є необхідним етапом в отриманні фаху з програмної інженерії. Перед викладачами, які викладають програмування, постає задача роз'яснити суть, переваги і недоліки ООП та навчити творчо і ефективно його застосовувати. При вирішенні даної задачі виникає проблема, яка полягає у тому, що ментальність сприйняття інформації у вітчизняних студентів не зовсім відповідає способу її подання в іноземній літературі з програмування, на основі якої в основному відбувається викладання. [2]. Крім того, у своїй більшості, вітчизняні та російськомовні підручники і посібники фактично повторюють методологію подання матеріалу зарубіжних джерел [3]. Основною відмінністю освоєння навчального матеріалу українськими студентами є критичне його сприйняття, як необхідна передумова для практичного застосування. Однак автори зарубіжних книжок (а за ними і вітчизняні автори) старанно уникають критичного підходу у своїх роботах. Можливо це пов'язано з тим, що крім подання інформації дані роботи виконують ще й рекламну функцію. Тому часто студенти не розуміють переваги й недоліки інструментів ООП та не знають, для яких задач, в яких випадках і як їх ефективно використовувати. При вивченні ООП одним з найбільш складних для студентів є поняття поліморфізму і механізмів його реалізації.

Метою даної статті є аспекти подання інформації про поліморфізм в ООП на основі критичного підходу який полягає у визначенні передумов і причин введення даної парадигми, її переваг і недоліків, аналізу проблем на цьому напрямку, розкриття механізмів поліморфізму та аналізу їх ефективності. Особлива увага приділяється віртуальним функціям, як найбільш складному поняттю для ефективного використання при програмуванні. Запропонована методологія була застосована автором у процесі викладання програмування в рамках дисциплін "Програмування", "Технології програмування", "Інженерія програмного забезпечення", "Прикладне програмування" [4]. Використання даної технології у навчальному процесі підтвердило її ефективність.

## Аспекти методології

Зазвичай вивчення ООП розпочинається з визначення трьох парадигм, що реалізує дана технологія: інкапсуляції, успадкування і поліморфізму. Проте, подальше викладання практично не пов'язане з цими парадигмами. Тому студенти, як правило, не розуміють значення такої інформації. Пропонується розпочати викладання даної теми з пояснення, що ООП було розроблено для ліквідації недоліків найбільш популярних на той час технологій програмування. Зростання розмірів розроблюваних програм виявили дві основних проблеми: по-перше, значно зросла кількість помилок у програмах, а по-друге, написання коду стало займати багато часу. Проте, існуючі в той час технології процедурного і структурного програмування перестали бути ефективними. для подолання цих проблем. Тому постала необхідність розробити нову технологію, яка б дозволила зменшити кількість помилок та використовувати раніше написаний код. Саме ці задачі і вирішує технологія ООП, яка базується на трьох принципах: інкапсуляції (для зменшення кількості помилок), а також успадкуванні і поліморфізму (для повторного використання коду). Успадкування використовується при конструюванні власних класів для прямого запозичення розробленого раніше коду, а поліморфізм призначений для адаптації запозиченого коду до умов його використання. Такий підхід дозволить студентам зрозуміти, що таке парадигми ООП і чому саме ці парадигми реалізовані.

Слід зазначити, що ООП має як переваги, так і недоліки. До недоліків можна віднести збільшення коду програм та часу їх виконання. Серед переваг ООП, крім вирішення вказаних раніше задач, можна відзначити можливість розпаралелювання процесу розробки програми, полегшення модифікації і рефакторингу коду та інші. Всі вони походять з того, що в даній технології програми розбиваються на окремі модулі – об'єкти, а людині властиво думати об'єктами. Саме це і обумовило популярність даної технології.

До вивчення механізмів поліморфізму потрібно переходити після вивчення того, як реалізована інкапсуляція, та після ознайомлення студентів з основними поняттями успадкування. Основна суть інкапсуляції полягає у тому, що вона вирішує задачу зменшення кількості помилок за рахунок закриття змінних класу і організації контрольованого доступу до них за допомогою функцій.

Успадкування вирішує задачу повторного використання коду за рахунок надання можливості при конструюванні нових класів використовувати як частини розроблені раніше класи, а не переписувати знову такий самий код.

Поліморфізм також призначений для вирішення даної задачі, оскільки він дозволяє зробити код, що буде повторно використовуватись, більш універсальним за рахунок незалежності від типів, які він обробляє. Проте, в мовах компіляторного типу повної незалежності коду від типів за допомогою ООП досягти не вдалося. Крім того, єдиного цілісного механізму поліморфізму не існує. Слід підкреслити, що принципи організації і механізми їх реалізації – це різні речі, між якими може не бути однозначого відображення. Один принцип може реалізуватись декількома механізмами, а деякі механізми можуть реалізувати не один, а декілька принципів. Наприклад, перевизначення функцій можна віднести як до успадкування, так і до поліморфізму. До основних механізмів реалізації поліморфізму у мові C++ можна віднести такі: перевантаження функцій, перевизначення функцій, віртуальні функції, інтерфейси, параметризовані типи. Дещо більші можливості поліморфізму мають мови C# [5] і Java [6].

Перевантаження функцій дозволяє ввести певну незалежність від типу функції. Критичний підхід до вивчення поняття перевантаження функцій полягає у роз'ясненні того, що на рівні реалізації перевантаження являє собою просто створення функцій різних типів з однаковим іменем, які потім у процесі компіляції отримують різні імена. Це полегшує розуміння даної технології.

Перевизначення функцій напряму створено для реалізації поліморфізму. Цей механізм полягає у тому, що у похідному класі створюється функція, яка має таку саму сигнатуру, як і функція базового класу, але інше тіло. Це дозволяє змінювати поведінку об'єктів похідних класів. Критичний підхід до вивчення поняття перевизначення функцій полягає у роз'ясненні того, що даний механізм не завжди коректно працює. Існують випадки, коли необхідно вказівнику на базовий клас присвоювати адресу об'єкта похідного класу. У таких випадках замість перевизначеної функції похідного класу буде помилково викликатись функція базового класу. Саме для ліквідації даного недоліку і були створені віртуальні функції, які об'являються у базових класах і реалізують відкладене зв'язування. Приклад реалізації поліморфізму за допомогою віртуальних функцій у C++:

```
class A
```

```
{
public:
    virtual void F{cout<<"A";}
}
class B:public A
{
public:
    void F(){cout<<"B";}
}
void main()
{
    A a *pa;
    B b;
    pa=&a;
    pa->F(); // виводиться A
    pa=&b;
    pa->F(); // виводиться B
}
```

Критичний підхід до вивчення поняття віртуальних функцій полягає у роз'ясненні того, що механізм відкладеного зв'язування потребує додаткових витрат часу і пам'яті. Тому віртуальною функцію роблять лише у тому випадку, коли у поточній програмі чи при повторному використанні коду потрібно буде змінювати тип об'єкта зв'язаного із вказівником, через який викликається дана функція. Певним недоліком віртуальних функцій у мові C++ є те, що один раз специфікована віртуальною у базовому класі дана функція залишається віртуальною у всій ієрархії успадкування, що іноді є недоцільним. У мові C# надано більше можливостей для роботи з віртуальними функціями у контексті поліморфізму. Дана мова програмування дозволяє керувати віртуальністю функцій не лише у базовому, але і у похідних класах. Для цього у похідному класі при перевизначенні віртуальної функції вказується одне з ключових слів "override" чи "new". Якщо вказано "override", то функція реалізує об'явлену у базовому класі віртуальність і передає її далі по їєрархії успадкування. Якщо ж вказано "new" чи не вказано нічого, то функція не реалізує віртуальність і не передає її по їєрархії успадкування. Приклад керування віртуальністю у мові C#:

```
class A
{
    public virtual void F(){Console.WriteLine("A")}
}
class B:A
{
    public override void F(){Console.Writeline("B");}
}
class C:A
{
    public new void F(){Console.WriteLine("C");}
}
class Program
{
    A a=new B();
    a.F(); // виводиться B
    a=new C();
    a.F();// виводиться A
}
```

Інтерфейси розширюють можливості поліморфізму. Фактично використання інтерфейсу є видом успадкування певної функціональності а реалізація функцій інтерфейсу подібна до їх перевантаження із врахуванням того, що віртуальність у цих функціях закладена за замовчуванням. Тому всі особливості реалізації поліморфізму тут повторюються. Перевага інтерфейсів у тому, що за допомогою них можна реалізувати поліморфізм не лише для класів, які зв'язані між собою

успадкуванням. Недоліком інтерфейсів є те, що їх не можна змінювати, оскільки це призведе до необхідності зміни коду у класах, що реалізують дані інтерфейси.

Параметризовані типи дозволяють реалізувати поліморфізм за рахунок відкладення конкретизації типів до етапу виконання. Вони широко використовуються при створенні бібліотек і дозволяють створювати код повністю незалежний від типів. Проте, повна незалежність від типу є також і недоліком, оскільки такі бібліотеки дозволяють виконувати лише ту обробку, яка дозволена у всіх типах. Для розширення можливостей повторного використання коду з параметризованими типами у мові C# введена можливість обмеження параметризованих типів. Отже, параметризовані типи потрібно використовувати лише в тих випадках, коли над ними виконуються дії, допустимі для кожної з подальших конкретизацій.

## Висновки

Мова програмування є інструментарієм для створення програм. Кожна з її технологій і кожен з механізмів є певними інструментами, які мають свої переваги і свої недоліки. Як правило, нові інструменти вводяться для ліквідації недоліків виявлених при використанні вже існуючих інструментів. Загострення уваги студентів не лише на перевагах, але також і на недоліках інструментів мови програмування дозволяє їм визначити межі можливостей даних інструментів і, таким чином, застерегти від неефективного їх використання. Це особливо важливо для технології ООП, оскільки неправильне використання даного підходу може значно погіршити код програм.

У статті подано лише окремі аспекти критичного підходу до викладання поліморфізму і взагалі ООП. Проте, даний підхід може бути розширений і використовуватись не лише для викладання поняття поліморфізму.

## СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ


1. Страуструп Б. Язык программирования C++. Специальное издание / Б. Страуструп : Пер. с англ. – М.: Издательство Бином, 2011 – 1136 с.
2. Эккель Б. Философия C++. Практическое программирование / Б. Эккель, Ч. Эллисон : Пер. с англ. – СПб. : Питер, 2004. – 608 с.
3. Павловская Т. А. C/C++. Программирование на языке высокого уровня / Т. А. Павловская – СПб. : Питер, 2002. – 464 с.
4. Азаров О. Д. Прикладне програмування : навч. посібник / О. Д. Азаров, О. І. Черняк, Л. А. Савицька – Вінниця : ВНТУ, 2016. – 131 с.
5. Троэлсен Э. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е изд. / Э. Троэлсен : Пер. с англ. – М. : ООО "И. Д. Вильямс", 2013. – 1312 с.
6. Дейтел Х. М. Технология программирования на Java 2 / Х. М. Дейтел, П. Дж. Дейтел, С. И. Сантри : Пер. с англ. – М.: ООО "Бином-Пресс", 2003 – 560 с.



***Олександр Іванович Черняк*** – канд. техн. наук, доцент кафедри обчислювальної техніки, Вінницький національний технічний університет, м. Вінниця.

***Oleksandr I. Chernyak*** – Cand. Sc. (Eng.), Assistant Professor of the Computer Techniques Chair, Vinnytsia National Technical University, Vinnytsia.