КОМП'ЮТЕРНІ СИСТЕМИ ТА КОМПОНЕНТИ

УДК УДК 004.432

О. Д. Азаров, О. І. Черняк, Л. А. Савицька

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

Вінницький національний технічний університет, м. Вінниця

Анотація. У статті запропоновано аспекти подання інформації про поліморфізм в ООП на основі критичного підходу, який полягає у визначенні суті даної парадигми, її переваг і недоліків, аналізу проблем на цьому напрямку, розкриття механізмів поліморфізму та аналізу їх ефективності. Актуальність даної статті пов'язана з тим, що українським студентам властиве критичне сприйняття інформації як необхідна передумова для ефективного практичного використання отриманих знань. 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 ефективність розробки програм. Дана методологія була успішно застосована автором у процесі викладання програмування в рамках дисциплін "Програмування", "Технології програмування", "Інженерія програмного забезпечення", "Прикладне програмування".

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

Аннотация. В статье описаны аспекты представления информации о полиморфизме в ООП на основе критического подхода, который заключается в определении сути данной парадигмы, ее преимуществ и недостатков, анализа проблем на этом направлении, раскрытие механизмов полиморфизма и анализа их эффективности. Актуальность данной статьи связана с тем, что украинским студентам свойственно критическое восприятие информации как необходимое условие для эффективного практического использования полученных знаний. Концентрация внимания студентов не только на преимуществах, но также и на недостатках инструментов языка программирования позволяет определить границы возможностей данных инструментов и, таким образом, предостеречь от неэффективного их использования. Это особенно важно для технологии ООП, так как неправильное использование данного подхода может значительно ухудшить эффективность разработки программ. Данная методология была успешно применена автором в процессе преподавания программирования в рамках дисциплин "Программирование", "Технологии программирования", "Инженерия программного обеспечения", "Прикладное программирование".

Ключевые слов: объектно-ориентированное программирование, методология преподавания, полиморфизм

Abstract. The article describes the aspects of the presentation of information about polymorphism in the OOP on the basis of a critical approach which consists in determining the essence of this paradigm, its advantages and disadvantages, analysis of problems in this direction, disclosure of mechanisms of polymorphism and analysis of their effectiveness. The urgency of this article is due to the fact that Ukrainian students have a critical perception of information as a prerequisite for the effective practical use of the knowledge gained. The intensification of students' attention not only on advantages but also on the disadvantages of programming language tools allows them to determine the limits of the capabilities of these tools and, thus, to prevent their ineffective use. This is especially important for OOP, because the misuse of this approach may significantly impair the effectiveness of program development. This methodology was successfully applied by the author in the course of teaching programming within the disciplines "Programming", "Programming Technologies", "Software Engineering", "Applied Programming". Using the proposed approach in the learning process has confirmed its effectiveness.

Key words: object-oriented programming, teaching methodology, polymorphism.

Вступ

Об'єктно-орієнтований підхід до побудови програмного забезпечення дозволив значно підвищити ефективність програмування, що в період розширення сфер використання обчислювальної техніки стало вирішальним чинником успішного впровадження програмних засобів у виробничій діяльності і побуті людини. З'явившись вперше у мові С++, даний підхід забезпечив настільки значні переваги на етапі розробки програм, що отримав назву об'єктно-орієнтованого програмування (ООП) і по суті став еталоном при створенні 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]. До того ж, 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 і через це не завжди 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.

Мета

Метою статті є аспекти подання інформації про поліморфізм в об'єктно-орієнтованому програмуванні на основі критичного підходу що 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 ООП. Дана методологія була застосована автором у процесі викладання програмування в рамках дисциплін "Програмування", "Технології програмування", "Інженерія програмного забезпечення", "Прикладне програмування" [4]. Використання запропонованого підходу у навчальному процесі підтвердило його 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). Інкапсуляція дозволяє зменшити кількість помилок за рахунок абстрагування від внутрішньої структури об'єктів. Успадкування використовується для запозичення розробленого раніше коду при конструюванні власних класів, а поліморфізм, зменшуючи типову залежність, дозволяє розширити повторне використання коду. Такий підхід дозволить студентам зрозуміти, що означають парадигми ООП і чому саме ці парадигми реалізовані.

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

До вивчення механізмів поліморфізму потрібно переходити після освоєння студентами поняття інкапсуляції, та ознайомлення їх з основами успадкування. Головна 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 оброблюваних типів. Проте, в мовах компіляторного типу неможливо досягти повної типової незалежності. Крім того, єдиного цілісного механізму поліморфізму не існує. Слід підкреслити, що принципи організації ООП 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 з однаковим іменем, але різними наборами параметрів. Потім у процесі компіляції дані функції отримують різні імена. Це полегшує розуміння

технології перевантаження.

Перевизначення функцій напряму створено для реалізації поліморфізму. Даний 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 такий механізм не завжди коректно працює. Існують випадки, коли вказівнику на базовий клас потрібно присвоювати адресу об'єкта похідного класу. У таких випадках через раннє зв'язування 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. Критичний підхід до викладання поняття інтерфейсів полягає у визначенні їхніх переваг і недоліків. 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 розробляти повністю незалежний від типів код. Проте, повна незалежність від типу є також і недоліком, оскільки такі бібліотеки повинні реалізувати лише ту обробку, що дозволена у всіх конкретизаціях. Недотримання цієї умови призводить до появи неконтрольованих помилок на етапі виконання. Для розширення можливостей повторного використання коду з параметризованими типами у мові C# введені так званні обмеження, що дозволяють на етапі компіляції виявляти помилки, викликані некоректною конкретизацією. 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, що мають свої переваги і 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 ефективність розробки 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 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.

Стаття надійшла: 25.08.2017.

Відомості про авторів

Азаров Олексій Дмитрович– док. техн. наук, професор кафедри обчислювальної техніки, Вінницький національний технічний університет.
Черняк Олександр Іванович– канд. техн. наук, доцент кафедри обчислювальної техніки, Вінницький національний технічний університет.
Савицька Людмила Анатоліївна, к. т. н., доцент кафедри обчислювальної техніки, ВНТУ, кафедра обчислювальної техніки.