УДК 53(07):004
DOI https://doi.org/10.32782/pet-2023-4-1

Вадим МУЛЯР
кандидат педагогічних наук, доцент кафедри експериментальної фізики, інформаційних та освітніх технологій, Волинський національний університет імені Лесі Українки, просп. Волі 13, м. Луцьк, Україна, 43025
ORCID ID: *https://orcid.org/0000-0003-4774-3947*
SCOPUS-AUTHOR ID: *58489973500*

Світлана ЯЦЮК
кандидат педагогічних наук, доцент кафедри загальної математики та методики навчання інформатики, декан факультету інформаційних технологій і математики, Волинський національний університет імені Лесі, просп. Волі 13, м. Луцьк, Україна, 43025
ORCID ID: *https://orcid.org/0000-0002-8369-6060*
SCOPUS-AUTHOR ID: *57221874892*

Валентина ЮНЧИК
старший викладач кафедри загальної математики та методики навчання інформатики, заступник декана з навчально-методичної роботи факультету інформаційних технологій і математики, Волинський національний університет імені Лесі, просп. Волі 13, м. Луцьк, Україна, 43025
ORCID ID: *https://orcid.org/0000-0003-3500-1508*
SCOPUS-AUTHOR ID: *57218347265*

Бібліографічний опис статті: Муляр, В., Яцюк, С., Юнчик, В. (2023). Методичні аспекти вивчення об'єктноорієнтованого програмування у закладах вищої освіти. *Фізика та освітні технології*, 4, 3–11, doi: https://doi.org/10.32782/pet-2023-4-1

МЕТОДИЧНІ АСПЕКТИ ВИВЧЕННЯ ОБ'ЄКТНООРІЄНТОВАНОГО ПРОГРАМУВАННЯ У ЗАКЛАДАХ ВИЩОЇ ОСВІТИ

У статті розкрито методичні аспекти формування професійних компетентностей майбутніх фахівців у галузі інформаційних технологій на основі об'єктноорієнтованого програмування. Акцентовано на тому, що ООП повністю відповідає звичному мисленню людини. На відміну від процедурно орієнтованого програмування ООП полегшує розробку та використання програм у випадку зростання коду зі збільшенням розміру проєкту, забезпечує приховування даних, дає змогу значно ефективніше імітувати події реального світу. Проаналізовано сучасний стан розробленості проблеми навчання об'єктноорієнтованого програмування у вищій школі. Встановлено, що у процесі вивчення ООП у здобувачів вищої освіти виникають труднощі, що пов'язані зі складністю об'єктноорієнтованої парадигми, їх здатності до розв'язання прикладних задач із застосуванням класів різних об'єктів та їх взаємодії. Запропоновано методику навчання ООП у закладах вищої освіти, в основу якої покладено розуміння здобувачами освіти фундаментальних понять і положень ООП, зв'язків між класами та об'єктами, формування у них об'єктноорієнтованого стилю програмування. Вивчення ООП розпочинають з розгляду сутності понять класу та об'єкту, розкриття відмінностей між ними. Зауважують, що клас може містити поля, методи, конструктори, блоки, вкладений клас та інтерфейс. Під час розгляду понять поля та методу підкреслюють, що метод використовують для повторного використання коду та його оптимізації. Розглядаючи види методів, особливу увагу приділяють користувацьким методам, їх опису та використанню. Підкреслюють, що статичний метод можна викликати без створення об'єкта, а абстрактний метод не має реалізації. Розглядаючи поняття конструктора, наголошують, що його призначення полягає в ініціалізації об'єкта. Після цього розкривають відмінності між конструктором та методом. Під час вивчення принципів ООП, наголошують на тому, що інкапсуляція дає змогу приховати дані та керувати доступом до них. Успадкування передбачає створення нових класів, які побудовані на наявних класах. Завдяки поліморфізму можна виконувати одну і ту ж дію різними способами. Абстракція дає змогу приховати деталі реалізації та показати суттєві властивості досліджуваного об'єкта чи явища. Наголошено, що ефективність навчання ООП значною мірою залежить від активізації навчально-пізнавальної діяльності студентів, їх уміння самостійно створювати проєкти із застосуванням об'єктноорієнтованого підходу.

Ключові слова: *парадигма, методологія, об'єктноорієнтоване програмування, компетентність, Java, клас, об'єкт, успадкування, інкапсуляція, поліморфізм.*

Vadim MULIAR
Candidate of Pedagogical Sciences, Associate Professor of the Department of Experimental Physics, Information and Educational Technologies of the Educational and Scientific Institute of Physics and Technology, Lesya Ukrainka Volyn National University, 13 Volya Ave., Lutsk, Ukraine, 43025
ORCID ID: *https://orcid.org/0000-0003-4774-3947*
SCOPUS-AUTHOR ID: *58489973500*

Svitlana YATSIUK
Candidate of Pedagogical Sciences, Associate Professor at the Department of General Mathematics and Methods of Teaching Computer Science, Lesya Ukrainka Volyn National University, 13 Volya Ave., Lutsk, Ukraine, 43025
ORCID ID: *https://orcid.org/0000-0002-8369-6060*
SCOPUS-AUTHOR ID: *57221874892*

Valentina YUNCHIK
Senior Lecturer at the Department of General Mathematics and Methods of Teaching Computer Science, Lesya Ukrainka Volyn National University, 13 Volya Ave., Lutsk, Ukraine, 43025
ORCID ID: *https://orcid.org/0000-0003-3500-1508*
SCOPUS-AUTHOR ID: *57218347265*

To cite this article: Muliar, V., Yatsyuk, S., Yunchyk, V. (2023). Metodychni aspekty vyvchennia obiektnooriientovanoho prohramuvannia u zakladakh vyshchoi osvity [Methodological aspects of learning object-oriented programming in higher education institutions]. *Physics and Educational Technology*, 4, 3–11, doi: https://doi.org/10.32782/pet-2023-4-1

METHODOLOGICAL ASPECTS OF LEARNING OBJECT-ORIENTED PROGRAMMING IN HIGHER EDUCATION INSTITUTIONS

The article reveals the methodological aspects of forming professional competences of future specialists in the field of information technology on the basis of object-oriented programming. It is emphasized that OOP fully corresponds to the usual human thinking. Unlike procedural-oriented programming, OOP facilitates the development and use of programs in the case of code growth with increasing project size, provides data hiding, and allows for much more effective simulation of real-world events. The current state of development of the problem of teaching object-oriented programming in higher education is analyzed. It is established that in the process of studying OOP, higher education students face difficulties associated with the complexity of the object-oriented paradigm, their ability to solve applied problems using classes of different objects and their interaction. A methodology for teaching OOP in higher education institutions is proposed, which is based on students' understanding of the fundamental concepts and provisions of OOP, the relationship between classes and objects, and the formation of an object-oriented programming style. The study of OOP begins with a consideration of the essence of the concepts of class and object, revealing the differences between them. It is noted that a class can contain fields, methods, constructors, blocks, a nested class, and an interface. When considering the concepts of field and method, it is emphasized that the method is used to reuse the code and optimize it. Considering the types of methods, special attention is paid to user methods, their description and use. It is emphasized that a static method can be called without creating an object, and an abstract method has no implementation. Considering the concept of a constructor, it is emphasized that its purpose is to initialize an object. After that, the differences between a constructor and a method are revealed. When studying the principles of OOP, it is emphasized that encapsulation allows you to hide data and control access to it. Inheritance involves creating new classes that are built on existing classes. Polymorphism allows you to perform the same action in different ways. Abstraction allows you to hide implementation details and show the essential properties of the object or phenomenon under study. It is emphasized that the effectiveness of teaching OOP largely depends on the intensification of students' learning and cognitive activities, their ability to create projects using an object-oriented approach.

Key words: *paradigm, methodology, object-oriented programming, competence, Java, class, object, inheritance, encapsulation, polymorphism.*

Актуальність проблеми. Парадигма об’єктноорієнтованого програмування (ООП) досить проста і розв’язує головну проблему – що робити зі складною предметною областю і складним кодом. Крім того, така парадигма ще й універсальна. Саме тому вона так добре прижилася.

ООП розглядає всю систему у вигляді об'єктів, які якимось чином один з одним взаємодіють. Оскільки людському мозку легше мислити об'єктами, ми автоматично розуміємо, що й у якого об'єкта має бути. Людині легко зрозуміти, де розташувати ті чи інші методи в коді. Завдяки цьому ООП забезпечує дуже легку і прозору структуру розташування коду. На сьогодні тільки об'єктноорієнтована парадигма є абсолютно універсальною. Переважна кількість завдань на ній вирішується максимально ефективно.

Об'єктноорієнтований підхід є найбільш зажаданий при розробці програмного забезпечення. Однак через дуже швидкий розвиток технологій та апаратно-програмних комплексів для формування у здобувачів освіти відповідних професійних компетенцій у цій сфері потрібно систематично переглядати та оновлювати підходи навчання об'єктноорієнтованого програмування.

Аналіз останніх досліджень і публікацій. Проблемі навчання об'єктноорієнтованого програмування й об'єктноорієнтованого підходу до розробки програмних продуктів присвячено дослідження вітчизняних науковців. С. Конюхов проаналізував сучасний стан розробленості проблеми професійної підготовки майбутніх інженерів-програмістів у процесі вивчення об'єктноорієнтованого програмування. Уточнив такі поняття, як «методологія ООП», «парадигма ООП», «технології ООП», «компетентність майбутнього інженера-програміста з ООП» (Конюхов, 2018). Науковець сформулював організаційно-методичні умови формування у випускників закладів вищої освіти компетентності з об'єктноорієнтованого програмування (Конюхов, 2019). Л. Зубик показала, що «Об'єктноорієнтоване програмування» є одним з тих фундаментальних курсів, вивчення яких повинно передувати опануванню фахових дисциплін (зокрема, «Web-технології та вебдизайн», «Організація баз даних і знань», «Технології комп'ютерного проєктування») і мати з їх змістом прямий теоретичний зв'язок (Зубик, 2016, с. 148–149). О. Теплицький дослідив дидактичні умови застосування об'єктноорієнтованого моделювання в підготовці майбутніх учителів інформатики (Теплицький, 2008; Теплицький, 2011). С. Лещук описала окремі кроки підготовки майбутніх фахівців, яких потребує ІТ-сфера. Науковець показала, що виконання студентами програмного проєкту сприяє розумінню принципів об'єктноорієнтованого програмування (Лещук, 2017, с. 84–85). О. Азаров, О. Черняк, Л. Савицька обґрунтували необхідність використання критичного підходу до викладання поняття поліморфізму в об'єктноорієнтованому програмуванні (Азаров, 2017). Аналіз доробок вітчизняних науковців дає підставу стверджувати, що об'єктноорієнтоване програмування є важливим елементом у формуванні професійної компетентності майбутніх фахівців у галузі інформаційних технологій.

Метою статті є розкриття методики навчання об'єктноорієнтованого програмування майбутніх фахівців у галузі інформаційних технологій.

Виклад основного матеріалу дослідження. Важливою складовою професійної компетентності майбутніх фахівців у галузі інформаційних технологій є компетентність з ООП, тобто «здатність розуміти фундаментальні основи об'єктноорієнтованого програмування, використовувати знання з царини методології та технології ООП й уміння виконувати декомпозицію та композицію предметної області, яка підлягає моделюванню, визначати властивості об'єктів і взаємодію між ними, розробляти алгоритми оброблення об'єктів, реалізовувати їх засобами об'єктноорієнтованих мов програмування для виконання професійних завдань і створення ефективних, надійних, якісних програм» (Конюхов, 2018, с. 169).

Як відомо, об'єктноорієнтований підхід до програмування ґрунтується на маніпулюванні об'єктами. Це означає, що розвиток логіки програми досягається шляхом визначення класів різних об'єктів та використання взаємодії об'єктів.

Аналіз наукових досліджень показує, що студенти, які до цього вивчали програмування хоча б на базовому рівні, гірше засвоюють ООП через зовсім інший процес мислення, ніж ті, які спочатку зрозуміли основи ООП, а потім вже почали розвивати знання з програмування. Це явище назвали «зміною парадигми». Виходячи з цього, робиться висновок про те, що вивчення ООП повинно передувати вивченню основ програмування. На думку авторів дослідження, це дасть змогу «легко зосередитися на поняттях

зв'язків класів і об'єктів замість того, щоб зосереджуватися на фактах структурного програмування» (Pasa Uysal, 2012).

Проте, результати деяких експериментальних досліджень показують протилежне – послідовність вивчення не має ніякого значення (Ehlert A., Schulte C., 2009).

Практика показує, що ефективність навчання ООП залежить також від вибору мови програмування, яка повинна володіти такими характеристиками, як кросплатформність, наявність різноманітних вільно поширюваних і комерційних середовищ розроблення, можливість використання для розв'язання широкого кола задач (Лещук, 2017, с. 48).

Виходячи з актуальності розв'язання наведених вище проблем, розглянемо методичні аспекти формування компетентності з ООП із використанням Java, яка є повністю об'єктноорієнтованою мовою програмування.

Вивчення ООП варто розпочати з огляду основних його понять та принципів. Зокрема, необхідно підкреслити, що ООП – це методологія або парадигма для розробки програм із використанням класів та об'єктів. Вона спрощує розробку та використання програмного забезпечення завдяки таким принципам, як-от: успадкування, поліморфізм, абстрагування, інкапсуляція тощо. Після розгляду основних принципів ООП, варто зауважити, що на відміну від процедурно орієнтованого програмування ООП полегшує розробку та використання програм у випадку зростання коду зі збільшенням розміру проєкту. Воно забезпечує приховування даних, тоді як у процедурно орієнтованій мові програмування до глобальних даних можна отримати доступ із будь-якого місця програми. ООП дає змогу значно ефективніше імітувати події в реальному світі. Після цього необхідно розглянути основні правила іменування Java, які полегшують читання коду програми. Підкреслити, якщо не дотримуватись цих правил, то це може призвести до плутанини або помилкового коду.

Під час вивчення теми «Об'єкти та класи» наголошують, що об'єкт у Java є фізичною, а також логічною сутністю, тоді як клас у Java є лише логічною сутністю. Сутність, яка має стан та поведінку, – це об'єкт, наприклад, маркер, стіл, автомобіль тощо. Вона може бути фізичною чи логічною (матеріальною та нематеріальною). Прикладом нематеріального об'єкта є банківська система. Об'єкт має такі характеристики, як-от: стан, що представляє дані (значення) об'єкта; поведінка, що представляє поведінку (функціональність) такого об'єкта; ідентичність, яка зазвичай реалізується за допомогою унікального ідентифікатора.

Зауважують, що класом є група об'єктів, які мають спільні властивості. Це шаблон або план, за яким створюються об'єкти. Це логічна сутність. Клас не може бути фізичним. Клас у Java може містити поля, методи, конструктори, блоки, вкладений клас та інтерфейс.

Далі вказують на те, що змінна, яка створюється всередині класу, але поза методом, називається полем. Під поле не виділяється пам'ять під час компіляції. Для нього виділяється пам'ять під час виконання, коли створюється об'єкт або екземпляр класу. Тому поле називають змінною екземпляра. Звертають увагу, що у Java метод подібний до функції, яка використовується для опису поведінки об'єкта. Застосування методу у програмі веде до повторного використання коду та його оптимізації.

Після детального розгляду різних способів створення та ініціалізації об'єкта, вводять поняття анонімного об'єкта та переходять до його використання в Java.

У наступній темі «Метод у Java» розкривають сутність поняття методу, розглядають типи методів, їх оголошення та виклику. Зауважують, що під методом розуміють блок коду або набору інструкцій, які згруповані для виконання певного завдання або операції. Його використовують для досягнення повторного використання коду. Метод також забезпечує легку зміну та читаність коду, просто додаючи або видаляючи частину коду. Він виконується лише тоді, коли ми його викликаємо. Декларація методу містить інформацію про атрибути методу, такі як видимість, тип повернення, ім'я та аргументи. При цьому варто докладно розглянути, яких типів можуть бути специфікатори доступу, які межі їх застосування. Звертають увагу, що у Java є два типи методів, а саме: стандартні та користувацькі. Розглядають призначення вбудованих методів та приклади їх використання. Особливу увагу приділяють вивченню користувацьких методів. Підкреслюють, що перш ніж викликати метод екземпляра, необхідно створити об'єкт його класу. Звертають увагу, що

існують два типи методів екземпляра: метод доступу та метод модифікатора.

Розглядаючи призначення та використання статичних методів, підкреслюють, що основною перевагою статичного методу є те, що ми можемо викликати його без створення об'єкта. За допомогою цього методу можна отримати доступ до статичних полів даних, а також змінити їх значення. Він використовується для створення методу екземпляра. Метод викликається за допомогою назви класу. Підкреслюють, що найкращим прикладом статичного методу є метод main().

Розкриваючи сутність абстрактного методу, зауважують, що метод, який не має тіла методу, називається абстрактним методом. Іншими словами, метод без реалізації є абстрактним методом. Він завжди оголошується в абстрактному класі. Це означає, що сам клас повинен бути абстрактним, якщо він має абстрактний метод. Для створення абстрактного методу ми використовуємо ключове слово abstract.

Вивчення теми «Конструктори» розпочинають з означення конструктора як блоку коду, що подібний до методу. Він викликається, коли створюється екземпляр класу. Під час виклику конструктора для об'єкта виділяється пам'ять. Зауважують, що це особливий тип методу, який використовується для ініціалізації об'єкта. Щоразу, коли об'єкт створюється за допомогою ключового слова new(), викликається принаймні один конструктор. Якщо в класі немає конструктора, то викликається конструктор за замовчуванням. У такому випадку компілятор Java надає конструктор за замовчуванням. Звертають увагу, що під час оголошення конструктора, потрібно дотримуватись певних правил.

Підкреслюють, що в Java є два типи конструкторів: конструктор без аргументів та конструктор з параметрами. Детально розглядають їх призначення та приклади використання. Зауважують, що конструктор у Java також можна перевантажити. Наводять приклади. Також розглядають способи копіювання значень одного об'єкта в інший.

На завершення вивчення теми розкривають відмінності між конструктором та методом у Java. Підкреслюють, що конструктор використовується для ініціалізації стану об'єкта, він не повинен мати тип повернення. Конструктор викликається неявно. Компілятор Java надає конструктор за замовчуванням, якщо у немає жодного конструктора в класі. Ім'я конструктора має бути таким же, як і ім'я класу. Метод використовується для виявлення поведінки об'єкта. Він повинен мати тип повернення. Метод викликається явно. Він у жодному разі не надається компілятором. Назва методу може збігатися, а може і не збігатися з назвою класу.

На початку вивчення теми «Ключове слово static у Java» зазначають, що це ключове слово в Java використовується головним чином для управління пам'яттю. Підкреслюють, що ключове слово static належить до класу, а не до екземпляра класу. Зауважують, що статичним може бути поле (змінна класу), метод, блок, вкладений клас. Далі переходять до детального розгляду застосування ключового слова static з полями, методами, блоками та вкладеними класами. Наводять приклади його використання.

Звертають увагу, що статична змінна може бути використана для посилання на загальну властивість усіх об'єктів (яка не є унікальною для кожного об'єкта), наприклад, на назву компанії співробітників, назву студентів коледжу тощо. Статична змінна отримує пам'ять лише один раз у зоні класу під час завантаження класу. Підкреслюють, що статичний метод належить класу, а не об'єкту класу. Його можна викликати без необхідності створення екземпляра класу. Статичний метод може отримати доступ до статичного поля і може змінити його значення. Зауважують, що статичний блок у Java використовується для ініціалізації статичного поля. Він виконується перед основним методом під час завантаження класу.

У темі «Ключове слово this у Java» докладно розглядають випадки використання у цього ключового слова в Java. Зокрема, це ключове слово можна використовувати для посилання на змінну екземпляра поточного класу. Якщо між змінними екземпляра та параметрами існує неоднозначність, ключове слово this розв'язує проблему двозначності. Також можна викликати метод поточного класу, використовуючи ключове слово this. Якщо ми не використовуємо це ключове слово, компілятор автоматично додає це ключове слово під час виклику методу. Окрім того, виклик конструктора this() можна використовувати для виклику конструктора поточного класу. Він використовується

для повторного використання конструктора. Підкреслюють, що ключове слово this також можна передати як аргумент у методі. В основному він використовується при обробці подій. Ми також можемо передати це ключове слово в конструктор. Це корисно, якщо нам доводиться використовувати один об'єкт у кількох класах. Ми можемо повернути ключове слово this як оператор методу. У такому випадку тип методу повернення повинен бути типом класу (не примітивним).

У темі «Інкапсуляція в Java», яка посідає важливе місце у вивченні ООП, підкреслюють, що під інкапсуляцією розуміють процес обгортання коду та даних в єдиний блок. Ми можемо створити повністю інкапсульований клас на Java, зробивши всі поля класу приватними. Потім використовувати методи setter і getter для встановлення та отримання даних. Це веде до контролю над даними. Інкапсуляція є способом приховати дані в Java, оскільки інші класи не зможуть отримати доступ до даних через приватні поля даних. Також інкапсульований клас легко перевірити. А це зручно у випадку модульного тестування. Окрім того, стандартні IDE дають змогу генерувати гетери та сетери. Це дозволяє легко і швидко створити інкапсульований клас. Переваги інкапсуляції в Java розглядають на конкретних прикладах.

Під час вивчення теми «Пакет у Java» зазначають, що під цим поняттям розуміють групу класів, інтерфейсів і підпакетів. Пакет можна розділити на два види: вбудований пакет і пакет, визначений користувачем. Існує багато вбудованих пакетів, таких як java, lang, javafx, net, io, util, sql тощо. Необхідно підкреслити, що пакет використовується у Java для групування класів та інтерфейсів, щоб їх можна було легко підтримувати. Він забезпечує захист доступу, усуває зіткнення імен. Далі розглядають приклади пакетів, як зібрати пакет та запустити програму з використанням пакетів та підпакетів. Також розкривають способи завантаження файлів класу або файлів jar.

У темі «Успадкування в Java» зазначають, що успадкування є тим механізмом, за якого один об'єкт набуває всіх властивостей та поведінки батьківського об'єкта. Підкреслюють, що успадкування є важливою частиною ООП. Ідея наслідування в Java полягає в тому, що ви можете створювати нові класи, побудовані на наявних класах. При успадкуванні від наявного класу можна повторно використовувати методи та поля батьківського класу. Крім того, можна додати нові методи та поля у поточний клас.

Далі зосереджують увагу на розкритті термінів, які використовуються в успадкуванні, таких як: клас; підклас/дочірній клас; суперклас/батьківський клас; повторне використання. Розглядають синтаксис та приклади успадкування в Java. Після цього зауважують, що у Java може бути три види успадкування: одно-, багаторівневе та ієрархічне. Пояснюють, що у випадку, коли клас успадковує інший клас, це називається однорівневим успадкуванням. Якщо існує ланцюг успадкування, це називається багаторівневим успадкуванням. А якщо два або більше класів успадковують один клас, це називається ієрархічним успадкуванням. Наводять приклади різних видів успадкування.

У наступній темі «Відносини між класами в Java» розглядають питання, як у програмі класи можуть бути пов'язані між собою. Зазначають, що IS-A відносини ґрунтується на успадкуванні класів або реалізації інтерфейсів. Наприклад, якщо клас Lorry розширює клас Car. У цьому випадку Lorry IS-A Car. Те саме стосується і реалізації інтерфейсів. Якщо клас Transport реалізує інтерфейс Moveable, то вони знаходяться у відношенні Transport IS-A Moveable. HAS-A відносини ґрунтується на використанні. Зауважують, що виділяють три варіанти відношення HAS-A: асоціація, агрегація та композиція. У цих відносинах асоціації об'єкти двох класів можуть посилатися один на одного. Наприклад, клас Horse HAS-A Halter, якщо код класу Horse містить посилання на екземпляр класу Halter. Звертають увагу, що агрегація та композиція є окремими випадками асоціації. Агрегація – відносини, коли один об'єкт є частиною іншого. А композиція – ще тісніший зв'язок, коли об'єкт не тільки є частиною іншого об'єкта, а й взагалі не може належати іншому об'єкту. Пояснюють, що різниця буде зрозуміла при розгляді цих відносин. І зупиняються на конкретних прикладах відношень між класами.

Вивчення теми «Перевантаження методу в Java» розпочинають з того, що пояснюють, що якщо клас має кілька методів з однаковими іменами, але різними за параметрами, то це називають перевантаження методів. Далі на

конкретних прикладах розглядають різні способи перевантаження методів у Java.

У наступній темі «Перевизначення методу в Java» розглядають випадки, коли підклас (дочірній клас) має той самий метод, що й оголошений у батьківському класі. Підкреслюють, що перевизначення методу використовується для забезпечення конкретної реалізації методу, який уже передбачений його надкласом. Іншими словами, метод перевизначення використовується для поліморфізму під час виконання. Зауважують, що при перевизначенні методів у Java потрібно дотримуватись наступних правил. Зокрема, метод повинен мати таку ж назву, що і в батьківському класі. Він повинен мати той самий параметр, що і в батьківському класі. Мають існувати відносини IS-A (успадкування). На завершення переходять до приладів перевизначення методу. Зокрема, пропонують розглянути ситуацію, коли банк є класом, який забезпечує функціональність для отримання процентної ставки. Однак процентна ставка залежить від банків. Наприклад, банки PB, IKIC та OXIS можуть надавати процентну ставку 9, 10 та 11%.

При вивченні поліморфізму наголошують, що це поняття, за допомогою якого можна виконувати одну і ту ж дію різними способами. Пояснюють, що в Java існує два типи поліморфізму: поліморфізм під час компіляції та поліморфізм під час виконання. Далі переходять до прикладів поліморфізму. Зокрема, можна розглянути завдання зі створення двох класів Bike та Splendor. Клас Splendor розширює клас Bike і замінює його метод run(). Ми викликаємо метод run за допомогою посилальної змінної класу Parent. Оскільки він посилається на об’єкт підкласу, а метод підкласу замінює метод класу Parent, метод підкласу викликається під час виконання.

Уводячи поняття абстрактного класу звертають увагу, що такі класи оголошуються за допомогою ключового слова abstract. Вони можуть містити абстрактні та неабстрактні методи (метод з тілом). Зауважують, що перед вивченням абстрактних класів варто ознайомитись з абстракцією. Пояснюють, що це процес приховування деталей реалізації та показ лише функціональності для користувача. Цей спосіб показує користувачеві лише важливі речі та приховує внутрішні деталі. Існує два способи досягнення абстракції в Java: нотація (від 0 до 100%); інтерфейс (100%). Далі підкреслюють, що абстрактний клас – це клас, який містить методи, що не мають реалізації. Він створюється з метою створення спільного інтерфейсу між різними реалізаціями класів, які будуть породжені від абстрактного класу. Абстрактний клас створюється для визначення деяких спільних рис класів, які будуть визначати конкретну реалізацію в породжених від нього класах.

Окрім того, повідомляють, що абстрактний клас має ряд особливостей: 1) він повинен бути оголошений за допомогою ключового слова abstract; 2) цей клас може містити абстрактні та неабстрактні методи; 3) заборонено (немає сенсу) створювати об’єкт абстрактного класу; 4) цей клас може містити як конструктори, так і статичні методи; 5) він може містити фінальні (final) методи, які змусять підклас не змінювати тіло методу.

Далі переходять до розгляду абстрактних методів, під якими розуміють такі методи, реалізація яких в програмі не має ніякого змісту. Підкреслюють, що абстрактний метод – це тільки оголошення форми (інтерфейсу), а не реалізація. Якщо у класі оголошено абстрактний метод, то клас також вважається абстрактним. Після цього на конкретних прикладах показують, що абстрактний клас може містити конструктор, поля та методи.

При вивченні інтерфейсу в Java звертають увагу на тому, що це проєкт класу. Повідомляють, що інтерфейс оголошується за допомогою ключового слова interface. Він забезпечує повну абстракцію. Це означає те, що всі методи в інтерфейсі оголошені з порожнім тілом, і всі поля є загальнодоступними, статичними та фінальними за замовчуванням. Клас, який реалізує інтерфейс, повинен реалізувати всі методи, оголошені в інтерфейсі. Далі наводять приклади інтерфейсів, розкривають зв’язок між класами та інтерфейсами. Після цього повідомляють, що за допомогою інтерфейсу можна підтримувати функціональність множинного успадкування. Розглядають приклади, в яких клас реалізує декілька інтерфейсів або інтерфейс розширює декілька інтерфейсів. На завершення з’ясовують відмінності між абстрактним класом та інтерфейсом, які демонструють на конкретних прикладах.

Висновки і перспективи подальших досліджень. Компетентність з ООП є важливою складовою професійної компетентності фахівців у галузі інформаційних технологій. На ефективність її формування впливають такі чинники, як складність об'єктноорієнтованої парадигми, послідовність вивчення ООП та основ програмування, вибір мови програмування та програмного забезпечення для навчання ООП. Запропоновано методику навчання ООП у закладах вищої освіти, в основу якої покладено формування у здобувачів освіти розуміння фундаментальних положень об'єктноорієнтованої парадигми, здатності до їх застосування при розв'язуванні прикладних задач. У процесі навчання ООП потрібно приділяти особливу увагу активізації навчально-пізнавальної діяльності студентів, їх самостійній роботі щодо проєктування класів, створення об'єктів, використання методів, забезпечення контролю доступу тощо. Перспектива подальших досліджень полягає у дослідженні застосування об'єктноорієнтованого підходу до розв'язання прикладних задач у різних предметних галузях.

ЛІТЕРАТУРА:

1. Азаров О.Д., Черняк О.І., Савицька Л.А. Аспекти критичного підходу до викладання поняття поліморфізму в об'єктноорієнтованому програмуванні. *Інформаційні технології та комп'ютерна інженерія.* 2017. Т. 39. № 2. С. 31–34.

2. Зубик Л.В. Формування професійних компетентностей майбутніх бакалаврів з інформаційних технологій у процесі вивчення фахових дисциплін: дис.… канд. пед. наук : 13.00.04 / Національний університет водного господарства та природокористування. Рівне, 2016. 341 с.

3. Конюхов С.Л. Організаційно-методичні умови формування професійної компетентності майбутніх інженерів-програмістів у процесі вивчення об'єктноорієнтованого програмування. *Фізико-математична освіта.* 2019. Випуск 4 (22). С. 68–74.

4. Конюхов С.Л. Професійна підготовка майбутніх інженерів-програмістів у процесі вивчення об'єктноорієнтованого програмування як проблема сучасної педагогічної науки. *Науковий вісник Мелітопольського державного педагогічного університету. Серія: Педагогіка.* 2018. № 1 (20). С. 166–172.

5. Лещук С.О. Окремі методичні аспекти підготовки ІТ-фахівців. *Інформаційні технології в освіті.* 2017. № 1(30). С. 81–96.

6. Теплицький О.І. Засоби навчання об'єктноорієнтованого моделювання студентів природничих спеціальностей педагогічних університетів. *Збірник наук. праць Кам'янець-Подільського нац. ун-ту. Серія педагогічна.* Кам'янець-Подільський : Кам'янець-Подільський нац. ун-т імені Івана Огієнка, 2011. Вип. 17 : Інноваційні технології управління компетентнісно-світоглядним становленням учителя: фізика, технології, астрономія. С. 246–248.

7. Теплицький О.І. Об'єктноорієнтоване моделювання в системі фундаменталізації підготовки майбутнього вчителя інформатики. *Збірник наукових праць. Педагогічні науки.* Вип. 50. Херсон : Видавництво ХДУ, 2008. Ч. 2. С. 285–288.

8. Ehlert A., Schulte C. (2009). Empirical comparison of objects-first and objects-later. *Proceedings of the fifth international workshop on Computing education research workshop.* 15–26. https://doi.org/10.1145/1584322.1584326

9. Pasa Uysal M. (2012) The Effects of Objects-First and Objects-Late Methods on Achievements of OOP Learners. *Journal of Software Engineering and Applications*. 5. 10. 816–822. http://dx.doi.org/10.4236/jsea.2012.510094

REFERENCES:

1. Azarov, O.D., Cherniak, O.I., Savytska, L.A. (2017). Aspekty krytychnoho pidkhodu do vykladannia poniattia polimorfizmu v obiektnooriientovanomu prohramuvanni [Aspects of a critical approach to teaching the concept of polymorphism in object-oriented programming]. *Informatsiini tekhnolohii ta kompiuterna inzheneriia.* 2. 31–34 [in Ukrainian].

2. Zubyk, L.V. (2016). Formuvannia profesiinykh kompetentnostei maibutnikh bakalavriv z informatsiinykh tekhnolohii u protsesi vyvchennia fakhovykh dystsyplin [Formation of professional competencies of future bachelors in information technology in the process of studying professional disciplines]: dys.… kand. ped. nauk: 13.00.04 / Natsionalnyi universytet vodnoho hospodarstva ta pryrodokorystuvannia. Rivne. 341 s. [in Ukrainian].

3. Koniukhov, S.L. (2019). Orhanizatsiino-metodychni umovy formuvannia profesiinoi kompetentnosti maibutnikh inzheneriv-prohramistiv u protsesi vyvchennia obiektnooriientovanoho prohramuvannia [Organizational and Methodological Conditions for the Formation of Professional Competence of Future Software Engineers in the Process of Learning Object-Oriented Programming]. *Fizyko-matematychna osvita.* 4 (22). 68–74 [in Ukrainian].

4. Koniukhov, S.L. (2018). Profesiina pidhotovka maibutnikh inzheneriv-prohramistiv u protsesi vyvchennia

obiektnooriientovanoho prohramuvannia yak problema suchasnoi pedahohichnoi nauky [Professional training of future software engineers in the process of studying object-oriented programming as a problem of modern pedagogical science]. *Naukovyi visnyk Melitopolskoho derzhavnoho pedahohichnoho universytetu. Seriia: Pedahohika.* 1 (20). 166–172 [in Ukrainian].

5. Leshchuk, S.O. (2017). Okremi metodychni aspekty pidhotovky IT-fakhivtsiv [Some methodological aspects of training IT specialists]. *Informatsiini tekhnolohii v osviti.* 1(30). 81–96 [in Ukrainian].

6. Teplytskyi, O.I. (2011). Zasoby navchannia obiektnooriientovanoho modeliuvannia studentiv pryrodnychykh spetsialnostei pedahohichnykh universytetiv [Tools for teaching object-oriented modeling to students of natural sciences at pedagogical universities]. *Zbirnyk nauk. prats Kamianets-Podilskoho nats. un-tu. Seriia pedahohichna.* Kamianets-Podilskyi : Kamianets-Podilskyi nats. un-t imeni Ivana Ohiienka, 17. 246–248 [in Ukrainian].

7. Teplytskyi, O.I. (2008). Obiektnooriientovane modeliuvannia v systemi fundamentalizatsii pidhotovky maibutnoho vchytelia informatyky [Object-oriented modeling in the system of fundamentalization of future computer science teacher training]. *Zbirnyk naukovykh prats. Pedahohichni nauky.* 50. 285–288 [in Ukrainian].

8. Ehlert A., Schulte C. (2009). Empirical comparison of objects-first and objects-later. *Proceedings of the fifth international workshop on Computing education research workshop.* 15–26. https://doi.org/10.1145/1584322.1584326

9. Pasa Uysal M. (2012). The Effects of Objects-First and Objects-Late Methods on Achievements of OOP Learners. *Journal of Software Engineering and Applications*. 5. 10. 816–822. http://dx.doi.org/10.4236/jsea.2012.510094