**МЕТОДИЧНА РОЗРОБКА**

для проведення практичного заняття з студентами 2-го курсу спеціальності 122 «Комп’ютерні науки» з дисципліни «Об’єктно-орієнтоване програмування»

**Лабораторне заняття:** «Застосування поліморфізму. Робота з методами: інкапсуляція»

**Мета:** закріпити теоретичні знання та практичні навики щодо перевизначення методів та конструкторів класів, а також написання рекурсивних методів при створенні прикладних програм за об’єктно-орієнтованим принципом. Закріплення навиків щодо застосування принципів поліморфізму в межах одного класу та між класами одного пакету. Закріпити теоретичні знання та практичні навики управління доступом до змінних екземплярів класів, а також застосування модифікаторів доступу **public**, **private** та **package**. Закріплення навиків щодо застосування принципів інкапсуляції в ієрархії класів.

**Час:** 2 годин

**Місце заняття:** комп’ютерна лабораторія

**Матеріальне забезпечення:**

- ПК з відповідним ПЗ та доступом до мережі Internet;

- Методичні рекомендації щодо виконання лабораторної роботи.

**Після лабораторного заняття студенти повинні:**

**вміти**: здійснювати перевизначення методів та конструкторів класу, створювати рекурсивні методи.

**знати:** принципи застосування поліморфізму та рекурсії в обє’ктно-орієнтованому програмуванні.

**План заняття:**

1. Перевизначення методів

2. Перевизначення конструкторів

3. Застосування модифікаторів доступу та методів get() і set()

4. Управління доступом в ієрархії класів

5. Видача індивідуальний практичних завдань

**1. Перевизначення методів**

Як зазначено в попередній лекції, перевизначення методів в межах одного класу можливо реалізувати шляхом заміни типу даних та кількості параметрів методу. Кількість параметрів, що передаються у метод можливо збільшувати, зменшувати, вводити, за їх відсутності, або видаляти у разі потреби. Відповідно під час виклику такого методу Java надаватиме можливість обрати необхідний варіант для виконання. Не слід забувати і про можливість перевизначення методу шляхом зміни типу даних, які повертає цей метод.

Розглянемо процедуру перевизначення методів детальніше. Для наочності обрано приклад розрахунку параметрів прямокутної пожежі (розглянутий на попередніх заняттях):

**public** **class** Firerectangular {

**double** length;

**double** width;

**public** Firerectangular(**double** length, **double** width) {

**this**.length = length;

**this**.width = width;

}

**double** square() {

**double** s;

s = length \* width;

**return** s;

}

**void** perimeter() {

**double** p = 2 \* (length + width);

System.***out***.println(p);

}

**void** front(**double** f, **double** n) {

**if** (f == length && n == 1) {

System.***out***.println(length);

} **else** **if** (f == length && n == 2) {

System.***out***.println(length \* 2);

} **else** **if** (f == width && n == 1) {

System.***out***.println(width);

} **else** **if** (f == width && n == 2) {

System.***out***.println(width \* 2);

} **else** {

System.***out***.println("Ви ввели не коректне значення");

}

}

}

В наведеному прикладі представлено клас Firerectangular із двома змінними екземпляра класу та конструктором для їх ініціалізації. В якості основних методів написані: типізований метод square() для визначення площі прямокутної пожежі, **void** -метод perimeter() – для визначення периметру та **void** -метод, що приймає параметри і визначає фронт пожежі front(**double** f, **double** n).

Зважаючи на мету заняття проведемо перевизначення означених методів в межах класу шляхом зміни типу даних з якими працюють методи та введенням параметрів у відповідні методи. Спочатку проведемо перевизначення методу square(). В якості параметрів методу передаються **локальні** змінні **int** length та **int** width, які **не залежать** від змінних екземпляра класу (локальні змінні перекривають змінні екземплярів класу з однаковими іменами):

**public** **class** Firerectangular {

**double** length;

**double** width;

**public** Firerectangular(**double** length, **double** width) {

**this**.length = length;

**this**.width = width;

}

**double** square() {

**double** s;

s = length \* width;

**return** s;

}

// перевизначений метод, що повертає значення площі пожежі

**int** square(**int** length, **int** width) {

**int** s = length \* width;

**return** s;

}

З наведеного прикладу спостерігається, що один і той самий метод square() може викликатись у двох різних варіантах виконання, а саме без прийняття параметрів із результатом роботи типу **double**, або із прийняття параметрів у вигляді локальних змінних **int** length і **int** width та результатом своєї роботи типу **int**.

Далі проведемо перевизначення методу perimeter() шляхом введенням параметру **double** n, який буде використано під час роботи методу:

**public** **class** Firerectangular {

**double** length; // змінні екземплярів класу

**double** width;

**public** Firerectangular(**double** length, **double** width) {

**this**.length = length;

**this**.width = width;

}

**double** square() {

**double** s;

s = length \* width;

**return** s;

}

// перевизначений метод, що повертає значення площі пожежі

**int** square(**int** length, **int** width) {

**int** s = length \* width;

**return** s;

}

**void** perimeter() {

**double** p = 2 \* (length + width);

System.***out***.println(p);

}

// перевизначений метод, що повертає знач. периметру пожежі

**void** perimeter(**double** n) {

**double** p = n \* (length + width);

System.***out***.println(p);

}

}

За умови перевизначення методу perimeter() користувач отримає можливість викликати його з main()-методу за умови введення аргументу **double** n, або без передачі аргументів (в такому випадку n = 2).

На черзі останній метод front() класу Firerectangular. Оригінальний метод відповідає за визначення фронту прямокутної пожежі залежно від однієї з чотирьох умов. Проведемо його перевизначення шляхом скорочення кількості параметрів, які передаються методу. Крім того логікою методу передбачено умову, що змінні екземпляра класу **double** length та **double** width рівні:

**public** **class** Firerectangular {

**double** length; // змінні екземплярів класу

**double** width;

**public** Firerectangular(**double** length, **double** width) {

**this**.length = length;

**this**.width = width;

}

**double** square() {

**double** s;

s = length \* width;

**return** s;

}

// перевизначений метод, що повертає значення площі пожежі

**int** square(**int** length, **int** width) {

**int** s = length \* width;

**return** s;

}

**void** perimeter() {

**double** p = 2 \* (length + width);

System.***out***.println(p);

}

// перевизначений метод, що повертає знач. периметру пожежі

**void** perimeter(**double** n) {

**double** p = n \* (length + width);

System.***out***.println(p);

}

**void** front(**double** f, **double** n) {

**if** (f == length && n == 1) {

System.***out***.println(length);

} **else** **if** (f == length && n == 2) {

System.***out***.println(length \* 2);

} **else** **if** (f == width && n == 1) {

System.***out***.println(width);

} **else** **if** (f == width && n == 2) {

System.***out***.println(width \* 2);

} **else** {

System.***out***.println("Ви ввели не коректне значення");

}

}

// перевизначений метод, що повертає значення фронту пожежі

// за умови рівності усіх сторін

**void** front(**double** n) {

**if** (n == 1) {

System.***out***.println(length);

} **else** **if** (n == 2) {

System.***out***.println(length \* 2);

} **else** **if** (n == 3) {

System.***out***.println(length \* 3);

} **else** {

System.***out***.println(length \* 4);

}

}

}

В результаті перевизначення методу front() буде отримано можливість визначення фронту пожежі за умови різних, або однакових значень змінних **double** length та **double** width.

Далі для тестування працездатності попередньо написаних та перевизначених методів проведемо їх виклик в головному класі з main()-методом:

**public** **class** MainFire {

**public** **static** **void** main(String[] args) {

Firerectangular fr = **new** Firerectangular(10, 10.5);

System.***out***.println(fr.square());

System.***out***.println(fr.square(5, 6));

fr.perimeter();

fr.perimeter(1.85);

fr.front(10, 1);

fr.front(3);

}

}

Результат роботи програми:

105.0

30

41.0

37.925000000000004

10.0

30.0

Доступний також варіант перевантаження одного і того ж методу в різних класах. Суть такого перевизначення полягає в тому, що для різних класів виклик методу з одним і тим самим іменем може виконувати різну логіку (спільний інтерфейс, логіка різна). Для наочності застосування такого інструменту створимо додатковий клас FireCircular, метою якого є визначення параметрів кругової пожежі:

**public** **class** FireCircular {

**double** radius;

**public** FireCircular(**double** radius) {

**this**.radius = radius;

}

}

Далі в означеному класі проведемо перевизначення методів square()та perimeter()таким чином, щоб назва, тип даних та параметри методів залишались не змінними. Наповнення методів (їх логіка) має визначати параметри пожеж різного виду, залежно від того до якого екземпляра класу їх буде викликано (параметри прямокутної або кругової пожежі):

**public** **class** FireCircular {

**double** radius;

**public** FireCircular(**double** radius) {

**this**.radius = radius;

}

**double** square() {

**double** s;

s = Math.***PI*** \* Math.*pow*(radius, 2);

**return** s;

}

**void** perimeter() {

**double** p = 2 \* Math.***PI*** \* radius;

System.***out***.println(p);

}

}

Для того, щоб переконатись в працездатності перевизначених методів square()та perimeter()в класі FireCircular, а також в тому, що перевизначення не вплинуло на відповідні методи класу Firerectangular проведемо їх виклик в класі MainFire із main()-методом:

**public** **class** MainFire {

**public** **static** **void** main(String[] args) {

Firerectangular fr = **new** Firerectangular(10, 10.5);

FireCircular fc = **new** FireCircular(15);

System.***out***.println(fr.square());

fr.perimeter();

System.***out***.println(fc.square());

fc.perimeter();

}

}

Результат роботи програми:

105.0

41.0

706.8583470577034

94.24777960769379

Як видно з наведеного прикладу виклик одного і того ж методу до різних екземплярів класу зумовлює до одержання різних результатів виконання. Це відбувається за рахунок виклику відповідного методу саме з того класу, до екземпляра якого застосовано виклик. Цей приклад яскраво відтворює принцип поліморфізму в об’єктно-орієнтованому програмуванні.

**2. Перевизначення конструкторів**

Зважаючи на те, що в процесі написання програм часто виникає необхідність ініціалізації змінних екземплярів класів по різному, в об’єктно-орієнтованих мовах програмування передбачено можливість перевизначення конструкторів. Перевизначення конструкторів також є однією із форм прояву поліморфізму. Для наочності використано попередній приклад класу Firerectangular із двома змінними екземплярів класу. В наведеному прикладі для ініціалізації змінних екземплярів класу було визначено стандартний конструктор, який надає можливість передати значення змінним під час створення екземпляра класу. Проте така форма не передбачає можливості ініціалізації змінних іншими способами у разі необхідності. Наприклад, якщо змінні **double** length та **double** width є рівними, то їх ініціалізація може проводитись з використанням одного параметру. Інший приклад це створення екземпляра класу без ініціалізації змінних (порожнього об’єкта), або передача в конструктор посилання на інший екземпляр класу з метою створення «клону» вже існуючого об’єкта.

В наступному прикладі проведемо процедуру перевизначення конструктора трьома різними способами. Кінцевий вигляд класу Firerectangular із перевизначеним конструктором матиме такий вигляд:

**public** **class** Firerectangular {

**double** length; // змінні екземплярів класу

**double** width;

// генерований конструктор

**public** Firerectangular(**double** length, **double** width) {

**this**.length = length;

**this**.width = width;

}

// перевизначений конструктор, яким передбачено

// встановлення рівності усіх сторін

**public** Firerectangular(**double** i) {

length = width = i;

}

// перевизначений конструктор, яким передбачено

// створення клону існуючого екземпляра класу

**public** Firerectangular(Firerectangular fr) {

length = fr.length;

width = fr.width;

}

// перевизначений конструктор, що не

// приймає жодного параметру

**public** Firerectangular() {

}

}

Далі в головному класі з main()-методом проведемо ініціалізацію змінних екземплярів класу із застосуванням усіх перевизначених конструкторів:

**public** **class** MainFire {

**public** **static** **void** main(String[] args) {

Firerectangular fr = **new** Firerectangular(10, 10.5);

Firerectangular fr1 = **new** Firerectangular(10);

Firerectangular fr2 = **new** Firerectangular(fr);

Firerectangular fr3 = **new** Firerectangular();

fr3.length = 11;

fr3.width = 12.5;

}

}

Як видно з представленого прикладу за умови виклику одного із перевизначених конструкторів користувачеві надається можливість ініціалізовувати змінні різними способами. В першому випадку проводиться стандартна ініціалізація, де за допомогою конструктора відбувається присвоєння значень зміним length та width. За умови застосування другого варіанту конструктора відбувається присвоєння однакового значення у дві змінні length та width. Третій конструктор надає можливість створити клон вже існуючого екземпляра класу ініціалізовуючи змінні ідентично до об’єкту, який передано в якості аргументу. Останній варіант конструктора надає можливість створювати об’єкти без ініціалізації змінних, а в разі виникнення такої потреби ініціалізація проводиться окремо.

**4. Застосування модифікаторів доступу та методів get() і set()**

Як зазначено в попередній лекції існують три типи модифікаторів доступу, а саме **public, private** та **package**. Останній модифікатор вважається модифікатором по замовчуванню та забезпечує доступ до елементів класу в межах пакету. Модифіктор **public** є відкритим та дозволяє доступатись до елементів помічених цим модифікатором з будь-якого класу (в межах Java-проекту). А модифікатор **private** захищає елементи класів від стороннього впливу та обмежує доступ лише в середині класу. Для доступу до закритих елементів (помічених як **private**) необхідно створювати спеціальні методи.

Для наочності застосування модифікаторів доступу розглянемо приклад попереднього практичного заняття щодо написання класу Firerectangular, який містить методи визначення параметрів прямокутної пожежі. Єдина відмінність – це модифікатори доступу до усіх методів позначимо за замовчуванням (без модифікаторів), а для змінних екземплярів класу модифікатор встановлено типу - **private**.

**class** Firerectangular {

**private** **double** length;

**private double** width;

Firerectangular(**double** length, **double** width) {

**this**.length = length;

**this**.width = width;

}

Firerectangular(**double** i) {

length = width = i;

}

Firerectangular(Firerectangular fr) {

length = fr.length;

width = fr.width;

}

Firerectangular() {

}

**double** square() {

**double** s;

s = length \* width;

**return** s;

}

**void** perimeter() {

**double** p = 2 \* (length + width);

System.***out***.println(p);

}

**void** front(**double** f, **double** n) {

**if** (f == length && n == 1) {

System.***out***.println(length);

} **else** **if** (f == length && n == 2) {

System.***out***.println(length \* 2);

} **else** **if** (f == width && n == 1) {

System.***out***.println(width);

} **else** **if** (f == width && n == 2) {

System.***out***.println(width \* 2);

} **else** {

System.***out***.println("Ви ввели не коректне значення");

}

}

}

У наведеному прикладі представлено декілька перевизначених конструкторів для ініціалізації змінних екземплярів класу. Ці конструктори є працездатними та дозволятимуть ініціалізовувати змінні при створенні екземплярів класу, адже знаходяться зі змінними типу **private** в межах одного класу. Проте для кращої наочності принципів інкапсуляції, при створенні екземпляру класу Firerectangular в main()-методі застосуємо порожній конструктор за прикладом:

**public** **class** MainFire {

**public** **static** **void** main(String[] args) {

Firerectangular fr = **new** Firerectangular();

}

Враховуючи, що в основному класі передбачений порожній конструктор такий спосіб оголошення екземпляра класу є цілком прийнятний. Далі, за умови застосування порожнього конструктора, необхідно ініціалізувати змінні екземпляра класу в main()-методі:

**public** **class** MainFire {

**public** **static** **void** main(String[] args) {

Firerectangular fr = **new** Firerectangular();

fr.length = 10; // ПОМИЛКА

fr.width = 20; // ПОМИЛКА

}

Такий спосіб ініціалізації не можливий, адже змінні length та width обмежені доступом лише з класу Firerectangular, а з класу MainFire цього зробити не можливо. Саме тому в класі Firerectangular (не зважаючи на наявні конструктори) необхідно передбачити методи, які дозволятимуть витягувати та встановлювати значення цих змінних з-за меж їх класу. Це методи get() і set() відповідно:

**public** **double** getLength() {

**return** length;

}

**public** **void** setLength(**double** length) {

**this**.length = length;

}

**public** **double** getWidth() {

**return** width;

}

**public** **void** setWidth(**double** width) {

**this**.width = width;

}

Ці методи можна прописувати в будь якій частині класу, проте рекомендовано їх висвітлювати в кінці класу. Методи getLength() та getWidth() призначені для отримання значень закритих змінних, а методи setLength() та setWidth() для встановлення значень цим змінним.

Розглянемо приклад застосування цих методів та використання ініціалізованих за їх допомогою змінних у класі MainFire:

**public** **class** MainFire {

**public** **static** **void** main(String[] args) {

Firerectangular fr = **new** Firerectangular();

fr.setLength (10);

fr.setWidth (20);

System.***out***.println(fr.getLength());

System.***out***.println(fr.getWidth());

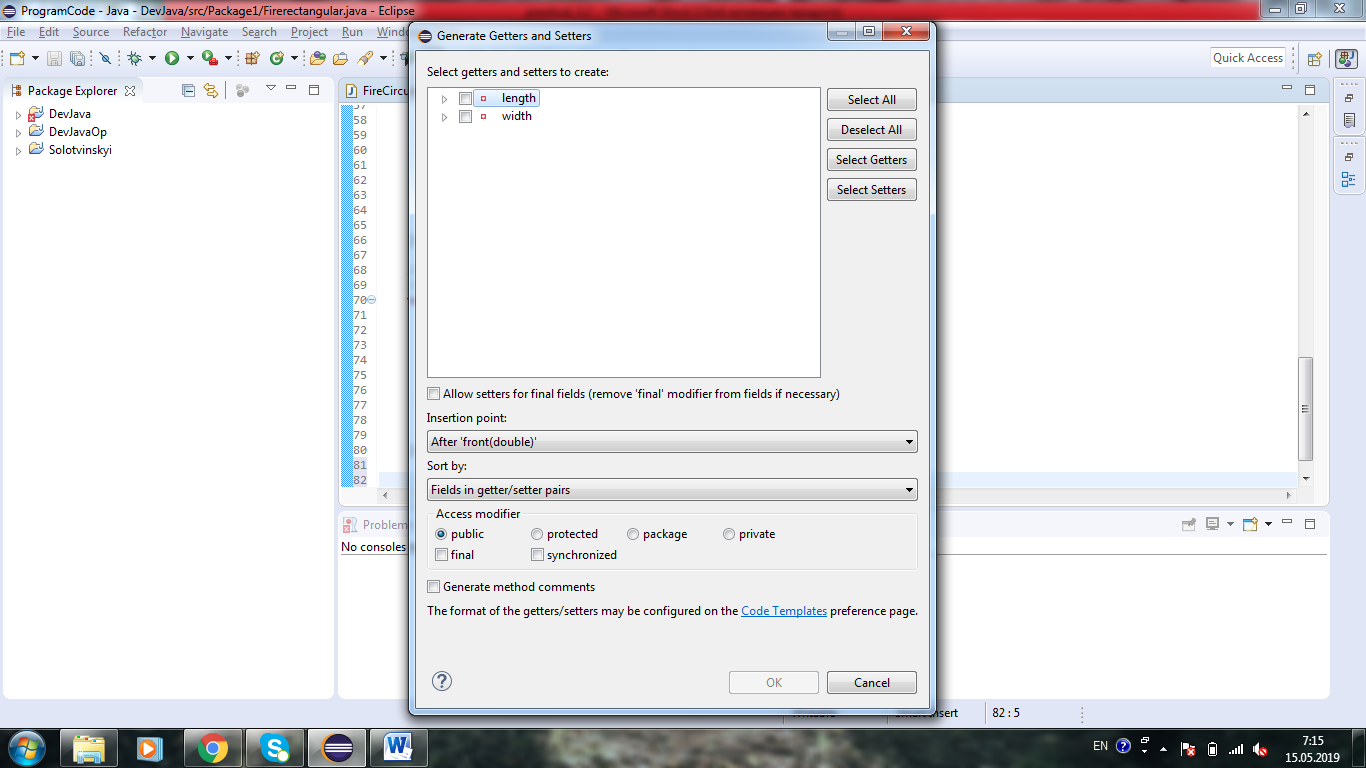
}

Результат роботи програми:

10

20

В середовищі розробки Eclipse передбачена можливість автоматичної генерації методів get() та set(). Для цього необхідно скористатись комбінацією клавіш Shift+Alt+S або меню Sourse та обрати Generate Getters and Setters. Після чого з’явиться контекстне меню де необхідно обрати відповідні змінні:



Слід відмітити, що в якості модифікаторів доступу для цих методів рекомендовано обирати **public** або **package**, адже доступ до цих методів має забезпечуватись з інших класів.

**2. Управління доступом в ієрархії класів**

Далі розглянемо приклад використання значень закритих змінних в методах інших класів. Для наочності скористаємось прикладом з попереднього практичного заняття, оберемо клас FireCircular, який містить методи для визначення параметрів кругової пожежі:

**class** FireCircular {

**private** **double** radius;

FireCircular(**double** radius) {

**this**.radius = radius;

}

**double** square() {

**double** s;

s = Math.***PI*** \* Math.*pow*(radius, 2);

**return** s;

}

**void** perimeter() {

**double** p = 2 \* Math.***PI*** \* radius;

System.***out***.println(p);

}

**void** front() {

**double** f = 2 \* Math.***PI*** \* radius;

System.***out***.println(f);

}

}

Як представлено в прикладі змінна radius помічена модифікатором доступу **private**, отже для неї необхідлно згенерувати методи get() та set():

**public** **double** getRadius() {

**return** radius;

}

**public** **void** setRadius(**double** radius) {

**this**.radius = radius;

}

Далі розглянемо новий клас FireAngular, який призначений для визначення параметрів кутової пожежі. Згідно існуючої методики, для визначення площі, фронту та периметру кутової пожежі необхідно знати радіус цієї пожежі. Для того, щоб не створювати ще одну змінну radius в класі FireAngular, ми можемо скористатись вже існуючою змінною **private** **double** radius з класу FireCircular (А тому б ні?). Отже розглянемо приклад класу FireAngular, який містить методи для визначення параметрів кутової пожежі та використовує значення змінної radius з класу FireCircular:

**public** **class** FireAngular {

**double** angular; // змінна екземплярів класу

// конструтор

**public** FireAngular(**double** angular) {

**this**.angular = angular;

}

// створення екземпляра класу FireCircular

FireCircular fc = **new** FireCircular(2);

//метод визначення площі пожежі

**double** square() {

**double** s = 0;

**if** (angular == 90) {

s = 0.25 \* Math.***PI*** \* Math.*pow*(fc.getRadius(), 2);

} **else** **if** (angular == 180) {

s = 0.5 \* Math.***PI*** \* Math.*pow*(fc.getRadius(), 2);

} **else** **if** (angular == 270) {

s = 0.75 \* Math.***PI*** \* Math.*pow*(fc.getRadius(), 2);

}

**return** s;

}

//метод визначення периметру пожежі

**void** perimeter() {

**if** (angular == 90) {

**double** p = (0.5 \* Math.***PI*** \* fc.getRadius()) + (2 \* fc.getRadius());

System.***out***.println(p);

} **else** **if** (angular == 180) {

**double** p = (Math.***PI*** \* fc.getRadius()) + (2 \* fc.getRadius());

System.***out***.println(p);

} **else** **if** (angular == 270) {

**double** p = (1.5 \* Math.***PI*** \* fc.getRadius()) + (2 \* fc.getRadius());

System.***out***.println(p);

}

}

// метод визначення фронту пожежі

**void** front() {

**if** (angular == 90) {

**double** f = 0.5 \* Math.***PI*** \* fc.getRadius();

System.***out***.println(f);

} **else** **if** (angular == 180) {

**double** f = Math.***PI*** \* fc.getRadius();

System.***out***.println(f);

} **else** **if** (angular == 270) {

**double** f = 1.5 \* Math.***PI*** \* fc.getRadius();

System.***out***.println(f);

}

}

}

В зазначеному прикладі є дві особливості. Перша це те, що в методах square(), perimeter() та front() використовується змінна radius доступ до якої забезпечується через метод getRadius(). Друга особливість – це те, що виклик методу getRadius() проведено з використанням екземпляра класу fc. Відповідно після ініціалізації конструктора класу FireAngular було створено екземпляр класу FireCircular з назвою fc.

Для перевірки того, що наведена конструкція має працездатний характер, проведемо виклик написаних методів до екземпляра класу FireAngular в main()-методі:

**public** **class** MainFire {

**public** **static** **void** main(String[] args) {

FireAngular fa = **new** FireAngular(180);

System.***out***.println(fa.square());

fa.perimeter();

fa.front();

}

}

Результат роботи програми:

6.283185307179586

10.283185307179586

6.283185307179586

Отже, як підсумок заняття, слід зазначити, що модифікатори доступу обмежують доступ до використання змінних екземплярів класу, що націлено на попередження несанкціонованих змін в роботі програми. Для доступу до закритих змінних екземплярів класу необхідно використовувати спеціальні методи. Основна суть принципу інкапсуляції полягає в управлінні доступом у ієрархії класів та пакетів, для чого в мовах програмування використовують модифікатори доступу.

**5. Видача індивідуальний практичних завдань**

У відповідності до тематики заняття для виконання практичного завдання необхідно розширити написану раніше об’єктно-орієнтовану програму (за результатами заняття №4) дотримуючись таких вказівок:

1. В кожному із 3-х існуючих класів програми (крім головного класу) необхідно:

1.1. Перевизначити щонайменше однин метод, що прийматиме в якості параметрів дані іншого типу із збереженням загальної логіки методу (наприклад: якщо дочірній метод працює з даними типу **int** то перевизначений метод має приймати в якості параметра та працювати з даними типу **double**);

1.2. Написати один метод з статичним ім’ям та динамічною логікою (назва, тип даних та параметри однакові, зміст різний) і перевизначити його таким чином, щоб за умови виклику цього методу до екземплярів різних класів в main()-методі результат виконання був унікальним (різним).

1.3. Усі модифікатори доступу **public** перевести у модифікатори по замовчуванню (**package** або без модифікатора);

1.4. Ісі змінні екземплярів класів (3-х класів) оголосити із модифікаторами доступу **private** та написати або згенерувати до кожної змінної методи get() i set().

1.5. Написати метод із довільною логікою, який в якості параметрів або внутрішніх змінних буде використовувати мінімум одну змінну екземплярів інших двох класів (наприклад для метода **class** Two необхідно використати змінні екземплярів класів **class** One та **class** Three тощо.)

2. В головному класі з main()-методом (4-й клас) викликати по одному новоствореному методу до відповідних екземплярів класів.

3. В головному класі з main()-методом (4-й клас) викликати усі перевизначені методи до відповідних екземплярів класів.

4. В одному із існуючих класів перевизначити конструктор 4-ма довільними варіантами таким чином, щоб в main()-методі, при створенні екземплярів цього класу, користувачеві надавалась можливість обирати 4 варіанти ініціалізації змінних. В головному класі з main()-методом ініціалізувати змінні усіма можливими варіантами при створенні екземплярів відповідних класів.

Результати виконання практичного завдання являються вихідними даними для виконання чергового індивідуального завдання у відповідності до програми курсу.

Звіт подається чотирма файлами із розширенням \*.java. Назви файлів мають відповідати назвам класів. Зміст файлів має відповідати поставленій умові в п.4. Програмний код рекомендовано супроводжувати коментарями щодо його змісту та логіки.

Звіт за результатами виконання лабораторної роботи (4 файли) завантажується у відповідну категорію Віртуального університету.