**МЕТОДИЧНА РОЗРОБКА**

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

**Лабораторне заняття 3.2:** «Створення власних класів та методів з довільною логікою»

**Мета:** здобути навики щодо написання власних класів та методів з довільною логікою (типізованих, void-методів та методів, що приймають параметри).

**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**: створювати класи, писати типізовані та void-методи із можливістю передачі їм параметрів.

**знати:** особливості взаємозв’язку між об’єктами, методами та змінними екземплярів класів.

**План заняття:**

1. Створення класів, екземплярів класів та змінних екземплярів класів

2. Створення конструкторів та ініціалізація змінних екземплярів класу з їх використанням

3. Написання власних типізованих, void-методів та методів, що приймають параметри.

4. Видача індивідуальних практичних завдань.

**1. Створення класів, екземплярів класів та змінних екземплярів класів**

Розпочнемо з основного – створення класу. Якщо приймати конструкцію класу абстрактно, то його можна порівняти з контейнером (оболонкою), в якій здійснюють написання програмного коду (усієї програмної логіки).

В якості наочного прикладу, розглянемо процедуру написання класу із визначення параметрів прямокутної пожежі. Створення класу розпочинається із ключового слова **class** та назви Firerectangular (модифікатор доступу **public** поки що не обов’язковий):

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

}

Після створення класу у відповідному каталозі комп’ютера (який вказано під час запуску середовища розробки) буде створено файл Firerectangular.java.

Наступним компонентом в ієрархії класу є його поля – змінні екземпляру класу. При оголошенні змінних зазначається тип даних та назва (аналогічно звичайним змінним):

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

 **double** length;

 **double** width;

}

Наступним кроком є створення екземпляра касу Firerectangular із довільним ім’ям. Екземпляри класів створюють в головному класі із main()-методом:

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

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

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

}

Враховуючи, що в основному класі Firerectangular оголошено дві змінні, то в класі MainFire, після створення екземпляра класу fr, необхідно їх ініціалізувати:

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

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

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

 fr.length = 10;

 fr.width = 10.5;

}

В результаті виконаних маніпуляцій змінні екземпляра класу fr (length та width) ініціалізовані та отримали значення 10 і 10,5 відповідно.

Один клас може мати довільну кількість екземплярів. І для кожного екземпляра класу, його змінні можуть ініціалізовуватись різними значеннями. Розглянемо приклад:

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

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

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

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

 fr.length = 10;

 fr.width = 10.5;

 fr1.length = 20;

 fr1.width = 21;

}

Далі, для визначення, наприклад, площі прямокутних пожеж для окремих екземплярів класів (fr та fr1), необхідно відтворити відповідні вирази із записом результатів розрахунків у новостворені змінні. Для виводу результатів у консоль скористаємось методом println():

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

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

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

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

 fr.length = 10;

 fr.width = 10.5;

 fr1.length = 20;

 fr1.width = 21;

 **double** s = fr.length \* fr.width;

 **double** s1 = fr1.length \* fr1.width;

 System.***out***.println("Площа першої пожежі: " + s);

 System.***out***.println("Площа другої пожежі: " + s1);

}

Недоліком подібного представлення коду є те, що за необхідності визначення площі пожежі для кожного екземпляру класу необхідно створювати свою підпрограму для розрахунку, а це значно нагромаджуватиме main()-метод, що є небежаним при об’єктно-орієнтованому підході. Якщо кількість екземплярів класів складатиме 20, то це зумовить до написання 20 окремих стрічок для розрахунку із оголошенням 20 нових змінних для запису результатів розрахунків. Це не зовсім практично та не відповідає парадигмі об’єктно-орієнтованого програмування. Методи, для виконання подібних розрахунків, мають володіти уніфікованим характером (написав 1 раз, використав n разів) та розміщуватись в основному класі. А main()-метод має забезпечувати лише їх виклик для конкретного екземпляру класу. Власне це питання буде більш детально розглянуто у наступних темах.

**2. Створення конструкторів та ініціалізація змінних екземплярів класу з їх використанням**

На даний момент ми навчились проводити ініціалізацію змінних екземплярів класу безпосередньо в main()-методі. За умови застосування такого підходу, кожну змінну потрібно інціалізовувати окремо із прив’язкою до визначеного екземпляру класу. Така процедура значно нагромаджує програмний код, наприклад:

fr.length = 10;

fr.width = 10.5;

fr1.length = 20;

fr1.width = 21;

Для вирішення цієї проблеми в програмуванні використовують **конструктори**. Згідно ієрархії програмного коду, конструктори в класі розміщуються після змінних екземплярів класу (хоча це не принципово). Розглянемо приклад:

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

 **double** length;

 **double** width;

 **public** Firerectangular(**double** l, **double** w) {

 length = l;

 width = w;

 }

}

Конструктор за формою схожий на типізований метод (буде розглянуто в наступній темі), який приймає параметри. Тільки в якості параметрів передаються значення змінних екземплярів класу, які присвоюються цим змінним в тілі конструктора за допомогою оператора « = ». Наявність конструктора значно спрощує ініціалізацію змінних екземплярів класу та дозволяє здійснити цю процедуру при першій згадці про екземпляр:

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

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

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

 Firerectangular fr1 = **new** Firerectangular**(20, 21)**;

}

Таким чином програмний код набуває значно естетичнішого вигляду та не нагромаджує main()-метод.

Більшість середовищ розробки володіють властивістю автоматичної генерації конструктора. До прикладу в IDE Eclipse, для доступу до цієї опції слід скористатись комбінацією клавіш Shift+Alt+S, після чого користувачеві надається можливість обрати перелік змінних, для яких генерується конструктор та місце розташування конструктора. Після автоматичної генерації конструктор набуватиме такого вигляду:

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

 **double** length;

 **double** width;

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

 **super**();

 **this**.length = length;

 **this**.width = width;

 }

}

За умови автоматичного генерування конструктора, спосіб ініціалізації змінних в головному класі залишається без змін. Як видно з представлених прикладів наявність конструктора значно спрощує процес ініціалізації змінних екземплярів класу, особливо за умови їх великої кількості.

**3. Написання власних типізованих, void-методів та методів, що приймають параметри.**

Як відомо, методи мають володіти уніфікованим характером (написав 1 раз, використав n разів) та розміщуватись в основному класі. А в main()-методі має забезпечуватись лише їх виклик для конкретного екземпляру класу. Саме тому, подальшу реалізацію методів необхідно виконувати дотримуючись зазначених рекомендацій.

Розпочнемо з типізованих методів. Ці методи працюють з визначеним типом даних та мають повертати результат того ж типу. Для цього в методі використовується ключове слово **return**. Для наочності розглянемо метод визначення площі прямокутної пожежі:

**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;

 }

}

Як видно з наведеного прикладу, уніфікований метод square() створюється в основному класі Firerectangular із прив’язкою до змінних екземплярів класу. Ключовими є два моменти – це тип даних, який зазначається перед назвою методу та слово **return** яке повертає значення того ж типу, який зазначено в оголошеному методі.

Далі з метою запуску методу, в головному класі слід виконати його виклик для визначеного екземпляру класу, що значно спрощує процедуру написання коду та не нагромаджує main()-метод:

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

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

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

 Firerectangular fr1 = **new** Firerectangular(20, 21);

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

 System.***out***.println(fr1.square());

}

Як видно з представленого прикладу, для розрахунку площі пожежі у двох випадках (для двох екземплярів класів) використовується один метод square(). Для чергового розрахунку площі пожежі, за умови створення нового екземпляру класу, не потрібно проводити повторних розрахунків, лише викликати вже існуючий метод square().

Методи **void**, на відміну від типізованих, не здійснюють жодних повернень (нічого не повертають). Методи цього типу в звичайному режимі виконують написаний в ньому програмний код (виконують логіку підпрограми). Після завершення виконання програмного коду, робота метода припиняється. Розглянемо приклад визначення периметру прямокутної пожежі шляхом написання **void**-методу perimeter():

**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);

 }

}

Проведемо тестування працездатності написаного методу в головному класі:

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

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

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

 Firerectangular fr1 = **new** Firerectangular(20, 21);

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

 System.***out***.println(fr1.square());

fr.perimeter();

fr1.perimeter();

}

Слід звернути увагу, що виклик методу perimeter() проведено без застосування методу println(), тому що він передбачений безпосередньо в тілі методу perimeter().

Наступний на черзі – це метод, що приймає параметри. До цього часу ми створювали не параметризовані методами square(),perimeter(). Це ті методи, після назви яких, дужки залишаються пустими (не приймають жодних параметрів). Відповідно за умови прийняття параметрів, метод може використовувати їх значення під час своєї роботи. Розглянемо приклад визначення фронту прямокутної пожежі (за умови розповсюдження одним або двома фронтами) front(**double** f, **double** n):

**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");

 }

 }

}

Є декілька особливостей роботи з параметризованими методами. Перше, це те, що тип даних приймаючих параметрів має відповідати типу даних з якими працює метод. Наступне – метод може примати декілька параметрів, за умови їх подання через знак «,». Ці методи можуть бути як типізованими, так і **void**-методами.

Спробуємо викликати написаний метод front(**double** f, **double** n) з головного класу:

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

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

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

 Firerectangular fr1 = **new** Firerectangular(20, 21);

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

 System.***out***.println(fr1.square());

fr.perimeter();

fr1.perimeter();

 fr.front(10, 1);

 fr1.front(21, 2);

 }

}

Як видно з прикладу, під час виклику методу front в дужках необхідно вказати значення параметрів (аргументів). Відповідно до вказаних аргументів програма виконує логіку та розрахунок використовуючи їх значення. Слід зауважити, що в самому методі front(**double** f, **double** n), який розміщено в класі Firerectangular, зазначаються *ПАРАМЕРИ* **double** f та **double** n, а при виклику методу fr.front(10, 1) в головному класі MainFire, вказуються вже *АРГУМЕНТИ* 10 та 1.

Отже, написання уніфікованих методів в тих класах, де необхідно застосовувати їх логіку, дозволяє виконувати їх виклик довільну кількість разів без необхідності повторного відтворення коду (логіки) методу.

**4. Видача індивідуальних практичних завдань**

У відповідності до тематики заняття для виконання індивідуального практичного завдання необхідно розширити написану об’єктно-орієнтовану програму (за результатами заняття №6.2) дотримуючись таких вказівок:

1. Створити три класи в межах одного пакету (наприклад: class One, class Two, class Three). Створення класів реалізується з допомогою меню File/New/Class в середовищі розробки IntellijIdea/Eclipse. При створенні класів не потрібно проводити автоматичне генерування main()-методу.

2. В кожному із створених класів оголосити по три змінні екземплярів класу із довільним типом даних без їх ініціалізації. Приклади оголошення змінних екземплярів класу наведено в методичній розробці, а також в лекції.

3. В кожному із трьох класів написати або згенерувати конструктор для спрощення процедури ініціалізації усіх змінних екземплярів класу.

4. В кожному із попередньо створених класів (наприклад: **class** One, **class** Two, **class** Three) написати:

4.1. **void**-метод з довільною логікою, який в процесі свого виконання використовуватиме значення змінних екземпляра класу;

4.2. Типізований метод (рекомендовано **double** або **int**) з довільною логікою, який в процесі свого виконання використовуватиме значення змінних екземпляра класу;

4.3. Обов’язкова умова: в одному із написаних класів створити метод, що приймає в якості параметру екземпляр іншого класу.

5. В головному класі з main()-методом (4-й клас) до кожного екземпляра класу викликати відповідні (усі написані в класі) методи на виконання.

Результатом виконання практичного завдання є розширена конструкція об’єктно-орієнтованої програми з довільною логікою, яка міститиме три класи із, щонайменше, двома методами, трьома змінними екземплярів класу та конструктором (в кожному класі). Результати виконання практичного завдання являються вихідними даними для виконання чергового індивідуального завдання у відповідності до програми курсу.