МЕТОДИЧНА РОЗРОБКА

для проведення лабораторного заняття з студентами (курсантами) 1-го курсу спеціальності 122 «Комп'ютерні науки» з дисципліни «Основи програмування»

Лабораторне заняття 6.4: «Створення власних методів з довільною логікою»

Мета: здобути навики щодо написання власних 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 (курсанти) повинні:
вміти: писати типізовані та void-методи із можливістю передачі їм параметрів.
знати: особливості взаємозв'язку між об'єктами, методами та змінними екземплярів класів.

Література:
1. Віртуальне навчальне середовище ЛДУ БЖД «Віртуальний університет» [Електронний ресурс]. – Доступний з : http://virt.ldubgd.edu.ua

План заняття:
1. Написання власних типізованих, void-методів та методів, що приймають параметри.
2. Видача індивідуальних практичних завдань.

1. 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.

2. 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. В кожному із попередньо створених класів (наприклад: `class One`, `class Two`, `class Three`) написати:
   - 1.1. `void`-метод з довільною логікою, який в процесі свого виконання використовуватиме значення змінних екземпляра класу;
   - 1.2. Типізований метод (рекомендовано `double` або `int`) з довільною логікою, який в процесі свого виконання використовуватиме значення змінних екземпляра класу;
   - 1.3. Обов'язкова умова: один із написаних методів (`void` або типізований) має приймати на вхід параметри, які будуть використані в процесі виконання логіки самого методу. В якості параметрів не рекомендовано використовувати змінні екземпляра класу.
   - 1.4. Не обов'язково, але дуже корисно: в одному із написаних класів створити метод, що приймає в якості параметру екземпляр іншого класу, а також поекспериментувати з методами, що приймають параметри динамічної довжини. Виконання цієї умови буде враховано при оцінюванні як виконання завдань підвищеної складності.
2. 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.

Звіт до індивідуального завдання подається чотирма файлами із розширенням *.java. Назви файлів мають відповідати назвам класів. Зміст файлів має відповідати визначеним вказівкам в п.2. Програмний код рекомендовано супроводжувати коментарями щодо його змісту та логіки, наприклад:

```
public class Firerectangular {
      // змінні екземплярів класу
      double length;
```

```
        double width;

        // генерований конструктор
        public Firerectangular(double length, double width) {
              super();
              this.length = length;
              this.width = width;
        }

        // типізований метод що повертає значення площі пожежі
        double square() {
              double s;
              s = length * width;
              return s;
        }
  }
```

Звіт за результатами виконання практичного завдання (4 файли) завантажується у відповідну категорію Віртуального університету.

Методичну розробку розробив:
Начальник кафедри УПІТтаТ
підполковник служби цивільного захисту Олександр ПРИДАТКО

Методична розробка обговорена на засіданні кафедри УПІТтаТ
Протокол № ____ від "___" _____ 20__ р.