ЗАТВЕРДЖУЮ
Завідувач кафедри управління проектами, інформаційних технологій та телекомунікацій, д.т.н., професор
Євген МАРТИН
«_ _»_____20___ р.

МЕТОДИЧНА РОЗРОБКА

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

Лабораторне заняття6.8: «Робота з методами: інкапсуляція»

Мета: 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 (курсанти) повинні:
вміти: організовувати управління доступом до змінних екземплярів класів за допомогою модифікаторів доступу та методів get() і set().
знати: принципи застосування інкапсуляції в обє'ктно-орієнтованому програмуванні.

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

План заняття:
1. Застосування модифікаторів доступу та методів get() і set()
2. Управління доступом в ієрархії класів
3. Видача індивідуальних практичних завдань

1. 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. Модифі-

ктор **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.

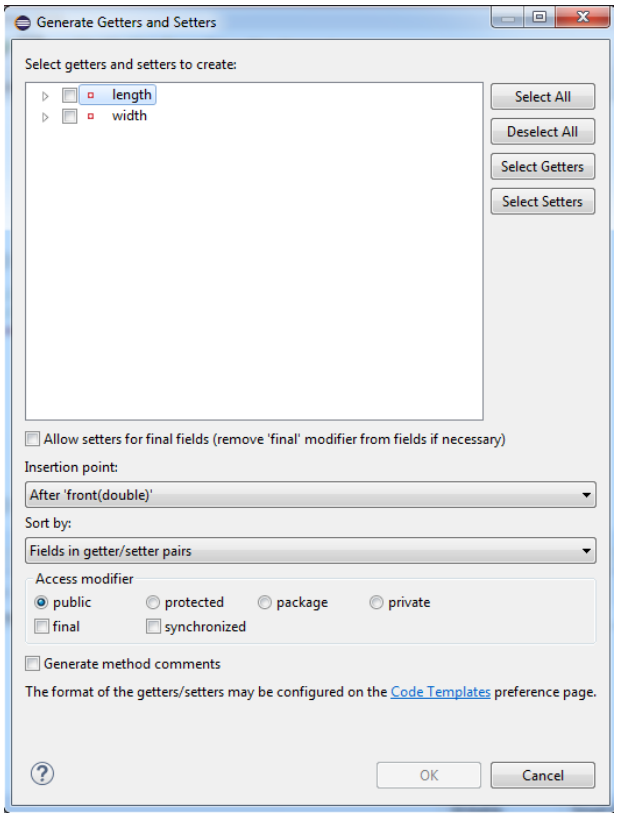
Розглянемо приклад застосування цих методів та використання ініціалізованих за їх допомогою змінних у класі `MainFire`:

```
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 необхідно 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.

3. Видача індивідуальних практичних завдань

У відповідності до тематики заняття для виконання практичного завдання необхідно розширити написану раніше об’єктно-орієнтовану програму (за результатами заняття №6.6) дотримуючись таких вказівок:

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

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

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

1.3. 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.

Результатом виконання практичного завдання є розширена конструкція об'єктно-орієнтованої програми з довільною логікою та дотриманням принципів інкапсуляції, яка міститиме методи, що використовують значення закритих змінних екземплярів інших 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.3. Програмний код рекомендовано супроводжувати коментарями щодо його змісту та логіки.

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

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

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