Кафедра управління проектами, інформаційних технологій та телекомунікацій

Спеціальність 122 «Комп'ютерні науки», ступінь вищої освіти «бакалавр»,
II семестр

Дисципліна «Основи програмування»

ТЕМА №6. ОСНОВИ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

ЛЕКЦІЯ №6.5. ПОЛІМОРФІЗМ

План:

Вступ. Введення в поліморфізм
1. Перевизначення методів
2. Перевизначення конструкторів

Література:

1. Java 8. Полное руководство. 9-е изд.: пер. с англ. / Герберт Шилдт. – Москва : ООО "И.Д. Вильяме", 2015. – 1376 с. ***(наявне в віртуальному середовищі)***

2. Head First Java (изучаем Java): пер. с англ. / Kathy Sierra, Bert Bates. – Москва : «Эксмо», 2012. – 718 с. ***(наявне в віртуальному середовищі)***

3. Програмування на Java (рос.) [Електронний ресурс]. – http://kostin.ws/java **(*наявне в віртуальному середовищі)***

4. Освоюємо Java. Вікіпідручник [Електронний ресурс]. – Доступний з https://uk.wikibooks.org/wiki/Освоюємо_Java ***(наявне в віртуальному середовищі)***

5. Java. ProgLang. Самоучитель (рос.) [Електронний ресурс]. – Доступний з http://proglang.su/java **(*наявне в віртуальному середовищі)***

Вступ. Введення в поліморфізм

Поліморфізм (від грец. «багато форм») – один із принципів ООП, який дозволяє використовувати спільний інтерфейс для загального класу дій, де кожна дія залежить від конкретної ситуації. Для прикладу розглянемо таку структуру даних як стек («останній поступив, перший опрацьований»). Припустимо, що для реалізації програми необхідно реалізувати три стеки: для цілочисельних типів даних, даних із плаваючою крапкою та для символів. Алгоритм реалізації усіх стеків є аналогічним, незважаючи на різні типи даних. Так от, якщо не використовувати принцип поліморфізму (створювати програму не за об'єктно-орієнтованим принципом) то для створення стеків необхідно було б реалізовувати три окремі програми з різними іменами, але аналогічною логікою. А використовуючи поліморфізм, для роботи з трьома стеками можливо створити одну підпрограму та двічі її перевизначити, при цьому назва

методу (підпрограми) у всіх трьох випадках буде ідентичною. Таким чином нівелюється необхідність розмножувати n кількість методів.

В більш загальному розумінні принцип поліморфізму можна виразити означенням «один інтерфейс, багато методів». Це означає, що можливо розробити один інтерфейс для групи зв'язаних дій. Такий підхід дозволяє спростити програму, оскільки спільний інтерфейс слугує для загального класу дій. А вибір конкретної дії (тобто метода) здійснюється окремо для конкретної ситуації та входить в обов'язки компілятора. Залучення компілятора звільняє програміста від необхідності здійснювати такий вибір вручну, йому необхідно лише пам'ятати про загальний інтерфейс та правильно його застосовувати.

Тепер для кращого розуміння розглянемо декілька прикладів. В першому прикладі задекларуємо, що собачий нюх – це поліморфна властивість. Якщо пес відчує запах кішки, то він рефлекторно почне гавкати. А у разі відчуття запаху їжі – відбуватиметься виділення слини. Отже, процес один – це нюх. Але залежно від умов де він застосовується, реалізується інша логіка. Відмінність в описаних ситуаціях, це предмет, який видає запах. На мові програмування ця відмінність полягає у різновиді параметрів та типів даних, які передаються методу для опрацювання.

Інший приклад це автомобіль. Поліморфізм відтворює здатність виробників автомобілів пропонувати більше різновидів автомобілів під однією маркою. До прикладу автомобіль може бути обладнаним звичайною гальмівною системою, або системою ABS, кермом із гідропідсилювачем або із звичайною рейкою, 4-х або 6-ти циліндрові двигуни тощо. Але в любому випадку для гальмування автомобіля необхідно натиснути на педаль гальм, обертати кермо аби здійснювати маневри та натискати на педаль акселератора аби збільшувати швидкість руху. Один і той самий інтерфейс може використовуватись для керування різноманітними реалізаціями.

Ознайомившись із загальними поняттями поліморфізму перейдемо до інструментів його реалізації.

1. Перевизначення методів

В Java дозволяється в одному класі визначати декілька методів із одним іменем, за умови якщо оголошення їх параметрів відрізняється, або відрізняється тип даних, що повертає метод. В такому разі методи отримують назву перевизначених. Перевизначення методів являється одним із способів підтримки поліморфізму в Java.

При виклику перевизначеного методу, для визначення його потрібного варіанту в Java використовують тип та (або) кількість аргументів методу. Відповідно перевизначені методи мають відрізнятись за типом та кількістю параметрів.

Типи перевизначених методів можуть відрізнятись, проте самого типу явно не достатньо для того, щоб відрізняти два різні варіанти методу. Коли в середовищі Java зустрічається виклик перевизначеного методу, то виконується

той варіант, параметри якого відповідають аргументам, що були вказані під час виклику метода.

Приклад перевизначення методу:

```
class Overload {
	// основний метод
	void test() {
		System.out.println("Параметри відсутні");
	}

	// перевизначений метод, перевірка наявності 1 числа int
	void test(int a) {
		System.out.println("a : " + a);
	}

	// перевизначений метод, перевірка наявності 2 чисел int
	void test(int a, int b) {
		System.out.println("a та b : " + a + " та " + b);
	}

	// перевизначений метод, перевірка наявності 1 числа double
	double test(double a) {
		return a;
	}
}

public class OverloadMain {

	public static void main(String[] args) {
		Overload ov = new Overload();

		ov.test();
		ov.test(5);
		ov.test(5, 10);
		System.out.println("a : " + ov.test(5.5));

	}
}

Результат роботи програми:
Параметри відсутні
a : 5
a та b : 5 та 10
a : 5.5
```

Як продемонстровано метод `test` було перевизначено чотири рази. Перший варіант взагалі не приймає параметрів, другий приймає один цілочисельний параметр, третій – два, а четвертий варіант приймає одне дробове число. З наведеного прикладу наочно видно, що компілятор Java

самостійно обирає на виконання той метод, параметри якого відповідають заданим аргументам.

Під час виклику перевизначеного метода відбувається співставлення типів даних аргументів, які використовуються для виклику метода, та типів даних параметрів самого метода. Проте співпадіння типів даних не завжди є обов'язковим. Інколи важливу роль в перевизначені методів може відігравати автоматичне приведення типів. Для наочності розглянемо наступний приклад:

```
class Overload {
	// основний метод
	void test() {
		System.out.println("Параметри відсутні");
	}
	// перевизначений метод, перевірка наявності 2 чисел int
	void test(int a, int b) {
		System.out.println("a та b : " + a + " та " + b);
	}
	// перевизначений метод, перевірка наявності 1 числа double
	double test(double a) {
		return a;
	}
}

public class OverloadMain {

	public static void main(String[] args) {
		Overload ov = new Overload();
		int i = 88;

		ov.test();
		ov.test(5, 10);

		// викликається варіант методу test (double)
		System.out.println("a : " + ov.test(5.5));

		// і тут викликається варіант методу test (double)
		System.out.println("a : " + ov.test(i));
	}
}

Результат роботи програми:
Параметри відсутні
a та b : 5 та 10
a : 88.0
a : 5.5
```

Як видно з наведеного прикладу, перевизначений метод `test(int)` не визначено. Тому при визові методу `test()` з цілочисельним аргументом в класі відсутній відповідний метод. Але Java здійснює автоматичне перетворення

типу `int` в `double`, щоб дозволити здійснити виклик необхідного варіанту методу. Так якщо варіант метода `test(int)` не буде визначено, тип змінної «і» автоматично приводиться до `double`, після чого викликається метод `test(double)`. Переконатись в цьому можливо просто, ініціалізація змінної проведена числом 88, а в результаті роботи програми ми отримали дробове значення 88,0. Звичайно якщо б варіант метода `test(int)` існував, то в описаному випадку проводився б виклик саме цього методу. Автоматичне приведення типів в Java проводиться лише тоді, коли не знайдено варіанту повного співпадіння.

Перевизначення методів підтримує поліморфізм, оскільки це один із способів реалізації принципу ООП «один інтерфейс, декілька методів». Проведемо роз'яснення цього тлумачення більш детальніше. В мовах програмування, що не підтримують принципів ООП, кожному новому методу має присвоюватись унікальне ім'я. Але найчастіше виникає потреба реалізації по сутності одного і того ж методу для різних типів даних. Розглянемо в якості прикладу функцію, яка приймає число по модулю. В мовах програмування, де перевизначення методів не підтримується, існує три або більше варіантів цієї функції. До прикладу в мові програмування С функція `abs()` повертає число по модулю із типом даних `integer`, функція `labs()` – значення типу `long integer`, функція `fabs()` – значення із плаваючою крапкою. В мові С перевизначення методів не підтримується, тому у кожної з вказаних функцій має бути унікальне ім'я, не дивлячись на те, що всі три функції виконують одну і ту задачу. На виході ситуація стає принципово складнішою, аніж могла бути. Хоча кожна функція побудована за одним принципом, програмісту на мові С необхідно пам'ятати три різні назви по суті одного і того ж методу. В об'єктно-орієнтованих мовах, які підтримують поліморфізм, така ситуація не виникає, оскільки усі методи визначення числа по модулю для різних типів даних можуть називатись однаково. В мові програмування Java метод `abs()` входить до стандартної бібліотеки та доступний через клас Math. В залежності від типу аргументу в Java викликається необхідний варіант методу.

Перевизначення методів цінне тим, що дозволяє звертатись до ідентичних методів за загальним іменем. Відповідно метод `abs()` представляє лише загальну дію, яка має виконуватись, а вибір відповідного метода покладається на компілятор.

При перевизначені методів, кожен його варіант має виконувати поставлені перед ним завдання. Не існує правила, відповідно до якого перевантажені методи мають бути зв'язані один з одним. Проте із стилістичної точки зору перевизначення методів передбачає їх певний зв'язок. Тому на практиці перевизначати слід тільки тісно пов'язані методи (за логікою та стилістикою).

Розглянемо ще один, не менш важливий приклад **перевантаження *методів, які виконують різну логіку в різних класах, проте носять спільне ім'я***. Такий спосіб потрібен для ідентифікації методів у різних класах, які виконують споріднену логіку (не ідентичну, а споріднену).

Припустимо маємо клас `Pet` в якому створено пустий метод `voice ()`:

```
public class Pet {

	public void voice ();
}
```

Після чого створюємо два класи Dog та Cat, які є дочірніми від класу Pet та наслідують усі його методи (наслідування вивчатимемо дещо пізніше, тому зараз особливої уваги на це поняття не звертаємо). Відповідно у кожному дочірньому класі метод voice() виконуватиме свою унікальну логіку, хоча носитиме спільне ім'я:

```
public class Dog extends Pet {

	@Override
	public void voice() {
		System.out.println("Я пес- Гауу-Гауу");
	}
}

public class Cat extends Pet{

	@Override
	public void voice() {
		System.out.println("Я кіт- Мяуу-Мяуу");
	}
}
```

Цікавим є те, що для наочності того, що метод є перевизначеним, перед ним генерується позначення @Override.

2. Перевизначення конструкторів

Крім перевизначення звичайних методів, перевизначеню піддаються і конструктори. Перевизначення конструкторів застосовується для надання можливості різної (уніфікованої) ініціалізації змінних екземпляра класу залежно від заданих умов. На практиці перевизначення конструкторів вважається нормою. Для наочності розглянемо приклад:

```
public class Box {
	double width;
	double height;
	double depth;

	public Box(double w, double h, double d) {
		width = w;
		height = h;
		depth = d;
```

```
	}

	double volume() {
		return width * height * depth;
	}
}
```

За такого варіанта виконання програми, під час створення екземпляра класу Box, компілятор запросить ініціалізувати відразу усі три змінні (задати аргументи для параметрів w, h, d).

```
Box myboxl = new Box (10,20,15);
```

З наведеного прикладу видно, що ініціалізація конструктора може проводитись лише так і ніяк інакше. При будь-якій іншій ініціалізації компілятор видаватиме помилку. А що робити, коли виникає необхідність не задавати жодного аргументу (пустий конструктор), або задати лише один аргумент, який автоматично буде ініціалізовано у всі три змінні екземпляру тощо. В такому випадку необхідно перевизначати конструктор:

```
public class Box {
	double width;
	double height;
	double depth;

	// конструктор для ініціалізації усіх розмірів
	Box(double w, double h, double d) {
		width = w;
		height = h;
		depth = d;
	}

	// конструктор коли не задається жодного розміру
	Box() {
		width = 0;
		height = 0;
		depth = 0;
	}

	// конструктор для створення куба (усі ребра рівні)
	Box(double len) {
		width = height = depth = len;
	}

	double volume() {
		return width * height * depth;
	}
}
```

```
public class BoxDemo {

      public static void main(String[] args) {
            Box mybox1 = new Box(5, 10, 15);
            Box mybox2 = new Box();
            Box mycube = new Box(7);

            System.out.println(mybox1.volume());
            System.out.println(mybox2.volume());
            System.out.println(mycube.volume());
      }
}

Результат роботи програми:
750.0
0.0
343.0
```

Як видно з прикладу, відповідний конструкт викликається залежно від аргументів, які вказано при створенні екземпляра класу.

Висновок: Як відомо найбільшої популярності за останні роки набирають саме об’єктно-орієнтовані мови програмування, які підтримують основні принципи цієї парадигми. Що стосується роботи з методами, то принципи поліморфізму є одними із найбільш вживаних, які дозволяють передбачати різні сценарії виконання однієї процедури не розмножуючи велику кількість методів з різними інтерфейсами. Розуміння цих положень надасть розробнику можливість створювати прості за архітектурою та високоефективні програмні коди.

Завдання на самопідготовку:

1. Опрацювати конспект лекції та підготуватись до складання тестових завдань.
2. Java 8. Полное руководство. 9-е изд.: пер. с англ. / Герберт Шилдт. – Москва : ООО "И.Д. Вильяме", 2015. – 1376 с. **(стор. 177-189)** ***(наявне в віртуальному середовищі)***
3. Head First Java (изучаем Java): пер. с англ. / Kathy Sierra, Bert Bates. – Москва : «Эксмо», 2012. – 718 с. **(стор. 214-226)** ***(наявне в віртуальному середовищі)***

Додатково:

4. Програмування на Java (рос.) [Електронний ресурс]. – http://kostin.ws/java **(розділи 9, 12) (*наявне в віртуальному середовищі)***
5. Освоюємо Java. Вікіпідручник [Електронний ресурс]. – Доступний з https://uk.wikibooks.org/wiki/Освоюємо_Java **(розділ 9)** ***(наявне в віртуальному середовищі)***

6. Java. ProgLang. Самоучитель (рос.) [Електронний ресурс]. – Доступний з http://proglang.su/java **(розділ 18) (*наявне в віртуальному середовищі)***