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 №6.3. ВВЕДЕННЯ В МЕТОДИ

План:

1. Типізовані та void-методи
2. Методи, що приймають параметри
3. Використання об'єктів в якості параметрів методу
4. Аргументи динамічної довжини

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)***

1. Типізовані та void-методи

Нам вже відомо низку методів, які володіють різноманітними можливостями та функціями. Зокрема, нами розглянуто реалізовані в Java методи класів `Math`, `String` тощо. Проте, основна суть програмування полягає у необхідності створення власних методів, які реалізують унікальну логіку, не передбачену стандартними методами.

Усі методи, які створюються розробниками програмного забезпечення володіють тією ж формою, що і метод `main()`, який широко використовувався в наведених раніше прикладах. Слід зауважити, що більшість методів, написаних власноруч, не оголошуватимуться як **`static`** (статичні) та **`public`** (публічними). Доречі, сам метод `main()` не є обов'язковим для загальної форми класу. Класи в Java можуть і не мати `main()` метода, він потрібен лише в тих випадках, коли

клас являється відправною точкою (запуском) для виконання усієї програми загалом, або для тестування окремих частин коду під час його написання.

Для того, щоб написати метод у класі потрібно застосувати загальну форму його оголошення, яка виглядає так:

```
тип імя_методу (список_парметрів) {
      // тіло методу
}
```

де `тип` означає визначений тип даних, який повертає (з якими працює) метод. Якщо метод не повертає жодного значення, то його типом має бути `void`. «Метод не повертає жодного значення» означає, що метод виконує певний перелік операцій визначений логікою програмного коду та не здійснює повернення результату своєї роботи (результат є, але методом не повертається).

За назву методу слугує його ідентифікатор `імя_методу`. А `список_параметрів` буде розглянуто в другому питанні лекції. Якщо в метода відсутні параметри то список параметрів залишається пустим (пусті дужки).

Далі розглянемо декілька реальних прикладів написання унікальних методів. Методи, що повертають значення після своєї роботи подаються з використанням оператора `return` за наступною формою:

```
int імя_методу () {
      int a = 5;
      int b = 5;
      int c = a + b;
      return c; // повертає результат роботи методу - 10
}
```

Методи, що не повертають значення подаються без оператора `return` за наступною формою:

```
void імя_методу () {
      int a = 5;
      int b = 5;
      int c = a + b;
      System.out.print (c); // в консоль буде виведено - 10
}
```

Для того, щоб викликати метод для визначеного об’єкту, необхідно зазначити назву об’єкта, та використовуючи оператор «.» здійснити виклик самого методу, наприклад:

```
об'єкт.імя_методу ();
```

Розглядаючи приклади з попередньої лекції, можна стверджувати, що клас `Box` міг би краще реалізовувати розрахунок об’єму, чим клас `BoxDemo`, адже не нагромаджуватиме `main()` – методу. Логічніше, якщо б такий

розрахунок виконувався в класі Box, оскільки об’єм залежить від розмірів, що представлені як змінні в тому ж класі Box. Для цього в клас Box необхідно ввести метод volume(). Приклад:

```
// файл Box.java
class Box {
      double width;
      double height;
      double depth;

      void volume() {  // метод розрахунку та виводу об’єму
            System.out.print("Об'єм становить ");
            System.out.println(width * height * depth);
      }
}


// файл BoxDemo.java
class BoxDemo {
      public static void main(String[] args) {

            Box myboxl = new Box();
            Box mybox2 = new Box();
            myboxl.width = 10; // ініціалізація змінних екземпляра
            myboxl.height = 20; // класу myboxl
            myboxl.depth = 15 ;

            mybox2.width = 3; // ініціалізація змінних екземпляра
            mybox2.height = 6; // класу mybox2
            mybox2.depth = 9;

            myboxl.volume ();//виклик методу для розрахунку 1 об’єму
            mybox2.volume ();//виклик методу для розрахунку 2 об’єму
      }
}
```

Результат:

```
Об'єм становить 3000.0
Об'єм становить 162.0
```

Розглянемо детальніше дві останні стрічки коду. В першій стрічці відбувається виклик методу volume() для об’єкту mybox1, а другій стрічці – для об’єкту mybox2. ***Особливість такого застосунку полягає в тому, що один і той самий метод, логіку якого викладено в класі, можна застосовувати для n кількості його екземплярів (об’єктів)***. За умови застосування подібного підходу нівелюється необхідність кожного разу при створенні нового об’єкту писати код для визначення його об’єму в main() – методі.

Враховуючи те, що метод volume () використовує значення змінних екземпляра класу до якого викликано метод, то результат роботи для двох

об’єктів отримано різний. Перевага такого застосунку очевидна: логіку методу, який написано один раз можна використовувати багато разів без його переписування.

Після виклику методу `volume()`, керуюча система Java передає керування коду визначеному в тілі методу `volume()`. Після закінчення виконання усіх операторів в тілі методу `volume()`, керування повертається до тієї частини програми де реалізувався виклик методу, після чого її виконання продовжується з наступної стрічки.

В методі `volume()` слід зауважити ще одну особливість: посилання на змінні екземпляра `width`, `height`, `depth` реалізовані без прив’язки до конкретного об’єкту.

Існує й інший спосіб представлення методу `volume()`, з обов’язковим поверненням значення, проте без виводу у консоль (на той випадок, коли потрібно визначити об’єм без його виводу на екран):

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

З наведеного прикладу може виникнути питання: як дістатися до значення, що повертає метод `volume()`? Для цього необхідно ввести додаткову змінну за прикладом:

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

double vol; // змінна яка приймає значення, що повертає метод
vol = mybox.volume ();
// …
```

Або цю процедуру можна виконати безпосередньо в самому методі `volume()`:

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

Наведені приклади є автентичними!

При роботі з методами, що повертають значення, слід пам’ятати:

1. Тип даних, який повертає метод має бути сумісним з типами, що використовуються у методі.

2. Змінна, що приймає значення роботи методу, також має бути сумісна з типом методу.

Узагальнення: усі данні, що використовуються в методі мають бути одного типу!

2. Методи, що приймають параметри

Нагадаємо приклад щодо форми написання методу:

```
тип імя_методу (список_парметрів) {
      // тіло методу
}
```

де `список_парметрів` означає послідовність пар «тип даних – назва», розділених комами. Параметри – це змінні, які приймають значення аргументів. Аргументи – це дані, що передаються методу під час його виклику. Іншими словами метод приймає параметри для їх опрацювання під час його виклику.

Методи, які приймають під час виклику параметри та виконують над ними певні операції називають ***параметризованими***. В якості прикладу розглянемо метод, що повертає квадрат числа 10 без прийняття параметрів:

```
int square ()  {
     return 10 * 10;
}
```

Недолік цього методу очевидний: він може визначати квадрат лише числа 10. А якщо потрібен метод піднесення до квадрату будь-якого числа? В такому разі слід написати метод із прийняттям відповідного параметру:

```
int square(int i) {
      return i * i;
}
```

Застосуємо даний метод для умовного екземпляра класу `two`:

```
…
two.square(2);
…

Результат роботи: 4.
```

Під час роботи з методами, які приймають параметри, слід розуміти два терміни: ***параметр*** та ***аргумент***.

Параметр – це визначена в методі змінна, яка приймає значення під час виклику методу. Наприклад в методі `square()` параметом є `i`.

Аргумент – це значення, яке передається методу під час його виклику. В наведеному прикладі аргументом є число 2.

Параметри зазначаються в дужках при написанні методу, а аргументи при його виклику!

Використовуючи методи з параметрами можна удосконалити розглянутий раніше клас Box. До цього моменту значення кожної змінної екземпляра класу необхідно було встановлювати індивідуально, використовуючи послідовність операторів:

```
myboxl.width = 10;
myboxl.height = 20;
myboxl.depth = 15;
```

Це працездатний код, проте через його громіздкий вигляд цілком реально допустити помилку, або не врахувати певного значення. Крім того в правильно написаних програмах на Java доступ до змінних екземпляра має реалізовуватись лише через методи, визначені в їх класі. В подальшому поведінку методу можна буде перевизначити, проте неможливо буде змінити поведінку змінної екземпляра.

Відповідно більш раціональним способом ініціалізації змінних (не беручи до уваги конструктори, які розглянуті на попередній лекції) є створення методу, який в якості параметрів приймає розміри та встановлює їх значення для кожної змінної екземпляра. Приклад:

```
// файл Box.java
class Box {
      double width;
      double height;
      double depth;

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

      void setBox(double w, double h, double d) {
            width = w;
            height = h;
            depth = d;
      }
}
// файл BoxDemo.java
class BoxDemo {
      public static void main(String[] args) {

            Box myboxl = new Box() ;
            Box mybox2 = new Box() ;
            myboxl.setBox(10, 20, 15);
            mybox2.setBox(3, 6, 9) ;

            System.out.println(myboxl.volume ());
            System.out.println(mybox2.volume ());
```

```
        }
    }
```

Як видно з наведеного прикладу, метод `setBox` використано для встановлення розмірів кожного параметру. Наприклад при виконанні стрічки `myboxl.setBox(10, 20, 15);` аргумент 10 присвоюється в параметр `w`, 20 – `h`, 15 – `d`. Після чого в тілі методу `setBox` значення параметрів `w, h, d` присвоюються змінним `width, height та depth` відповідно.

Так, метод `setBox()` повністю виконує функціонал конструкторів класу, які ми вже розглядали. Проте в даному випадку, такий приклад застосований для наочності процедури передачі параметрів для методу.

3. Використання об’єктів в якості параметрів методу

В попередніх прикладах у якості параметрів розглядались лише примітивні типи даних. Проте передача методам об’єктів в якості параметрів не лише доступна, але й доволі розповсюджена практика. Розглянемо приклад:

```
public class Test {
    int a, b;

 // створюємо конструктор
    Test (int i, int j){
        a = i;
        b = j;
    }

// передаємо об’єкт в якості параметру
    boolean equals (Test o){
        if (o.a == a && o.b == b) return true;
        else return false;
    }
}

public class PassOb {

    public static void main(String[] args) {
        Test ob1 = new Test(100, 22);
        Test ob2 = new Test(100, 22);
        Test ob3 = new Test(-1, -1);

        System.out.println("ob1 == ob2: " + ob1.equals(ob2));
        System.out.println("ob1 == ob3: " + ob1.equals(ob3));
    }
}
```

Результат роботи програми:

```
ob1 == ob2: true
ob1 == ob3: false
```

В наведеному прикладі метод `equals ()` перевіряє в класі `Test` на рівність два об’єкти та повертає логічний результат. Метод порівнює об’єкт, до якого його викликано із об’єктом, який переданий йому в якості аргументу. У випадку, якщо два об’єкти місять однакові значення метод повертає логічне значення `true`, в іншому випадку `false`.

Особливу увагу слід звернути на те, що в якості типу даних для параметра «o» в методі `equals ()` вказано клас `Test`. Ця процедура і відображає усю сутність передачі об’єктів в якості параметрів.

В якості параметрів об’єкти найчастіше зустрічаються в конструкторах. Інколи виникає необхідність створити новий екземпляр класу ідентичний до вже існуючому екземпляру (створити клон). Для цього необхідно визначити конструктор, що приймає в якості параметра об’єкт (екземпляр) свого класу. Інакше кажучи один об’єкт ініціалізується іншим об’єктом. Розглянемо приклад:

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

	// конструктор в якому параметр - це об'єкт типу Box
	Box(Box ob) {
		width = ob.width;
		height = ob.height;
		depth = ob.depth;
	}

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

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

public class BoxDemo {

	public static void main(String[] args) {
		Box mybox1 = new Box(5, 10, 15); // використання
стандартного конструтора
		Box myclone = new Box(mybox1); // створення клону
```

```
            System.out.println(mybox1.volume());
            System.out.println(myclone.volume());
      }
}

Результат роботи програми
750.0
750.0
```

Почавши розробляти власні класи, приходить розуміння того, що на «озброєнні» необхідно мати декілька форм конструкторів. Вони дозволяють зручно та ефективно створювати потрібні екземпляри класів. Це питання більш детальніше буде розглянуто в наступних темах (поняття поліморфізму).

Для передачі аргументів методу існує ДВА способи: виклик за значенням та виклик за посиланням. Коли методу передається аргумент примітивного типу, його передача відбувається за значенням. Як наслідок створюється копія аргументу та всі зміни, що проводяться в подальшому із параметром, який приймає цей аргумент, не завдають жодного впливу за межами викликаного методу. В якості прикладу розглянемо наступну програму:

```
//аргументи примітивних типів передаються за значенням

public class TestMethod {
      void meth (int i, int j){
            i *= 2;
            j /=2;
      }
}


public class TestMain {
      public static void main(String[]args) {
            TestMethod ob = new TestMethod();

            int a = 15, b = 20;
            System.out.println("a та b до виклику: " + a + " " + b);

            ob.meth(a, b);
            System.out.println("a та b після виклику: " + a + " " +
b);
      }

}

Результат роботи програми:
a та b до виклику: 15 20
a та b після виклику: 15 20
```

Як видно з наведеного прикладу, операції, що прописані в методі `meth()`, не вказують жодного впливу на значення змінних a та b, які використовуються при виклику цього методу. Їх значення не змінилось на 30 та 10 відповідно.

При передачі об'єкта в якості аргументу для метода, ситуація змінюється кардинально, оскільки об'єкти, за своєю сутністю, передаються при виклику за посиланням. При оголошенні змінної типу класу (посилкового типу) створюється лише посилання на об'єкт цього класу. Таким чином, при передачі цього посилання методу, приймаючий її параметр буде посилатись на той самий об'єкт, на який посилається аргумент. Об'єкти діють так, наче вони передаються методам за посиланням. Будь які зміни об'єкту в тілі методу вказують вплив на об'єкт, який передано в якості аргументу. Для наочності розглянемо приклад:

```
public class TestMethod {
	int a, b;

	TestMethod(int i, int j) {
		a = i;
		b = j;
	}

	// передаємо об'єкт
	void meth(TestMethod o) {
		o.a *= 2;
		o.b /= 2;
	}
}

public class TestMain {
	public static void main(String []args ){
		TestMethod ob = new TestMethod(15, 20);

		System.out.println("a та b до виклику: " + ob.a + " " +
ob.b);

		ob.meth(ob);
		System.out.println("a та b після виклику: " + ob.a + " "
+ ob.b);
	}
}

Результат роботи програми:
a та b до виклику: 15 20
a та b після виклику: 30 10
```

Як видно з наведено прикладу, дії, які виконуються в тілі методу `meth()`, вказують вплив на об'єкт, що вказано в якості аргументу.

Як відомо, типізовані методи можуть повертати значення будь-якого типу даних. Це також стосується і посилкових типів (типів певного класу). В

наведеному далі прикладі метод `incrByTen ()` повертає об’єкт в якому значення змінної збільшується на 10 у порівнянні з переданим аргументом.

```
public class TestMethod {
	int a;

	TestMethod(int i) {
		a = i;
	}

	TestMethod incrByTen() {
		TestMethod temp = new TestMethod(a + 10);
		return temp;
	}
}

public class TestMain {
	public static void main(String []args) {
		TestMethod ob1 = new TestMethod(2);
		TestMethod ob2;

		ob2 = ob1.incrByTen();
		System.out.println("ob1.a: " + ob1.a);
		System.out.println("ob2.a: " + ob2.a);

		ob2 = ob2.incrByTen();
		System.out.println("ob2.a після повторного збільшення: "
+ ob2.a);
	}
}

Результат роботи:
ob1.a: 2
ob2.a: 12
ob2.a після повторного збільшення: 22
```

Як видно при кожному виклику метода `incrByTen ()` створюється новий об’єкт, а посилання на нього повертається тій частині програми де відбувається виклик. В наведеному прикладі спостерігається ще один цікавий момент. Пам'ять виділяється для всіх об’єктів динамічно з допомогою оператора new, а відповідно розробнику не потрібно приймати жодних мір, для того, щоб об’єкт не вийшов за межі області своєї роботи. Це стало можливо за рахунок того, що виконання методу, де створюється об’єкт, обов’язково закінчує свою роботу. Об’єкт буде існувати до тих пір, доки буде існувати посилання на нього в будь-якому іншому місці програми. А за відсутності будь-яких посилань на об’єкт він буде знищений при наступному збиранні «сміття».

4. Аргументи динамічної довжини

Метод, що приймає динамічну кількість аргументів (кожного разу іншу кількість) має назву методу з аргументами динамічної довжини. Ситуації коли методу необхідно передати динамічну кількість аргументів зустрічаються не часто, проте зустрічаються! До прикладу, метод, призначення кого встановлення з'єднання з Інтернетом, може приймати ім'я користувача, ім'я файлу, пароль, мережевий протокол та низку інших даних. Проте якщо певні дані не вказано, методи можуть обирати значення за замовчуванням. В подібній ситуації було б зручно передавати лише ті аргументи, до яких не застосовано значень за замовчуванням.

Для зазначення аргументів динамічної довжини слугує оператор «…». В наведеному нижче прикладі продемонстровано, яким чином можливо оголосити метод з аргументами динамічної довжини^

```
static void vaTest(int... v)
```

За такої синтаксичної конструкції компілятору передається, що метод `vaTest()` може викликатись без аргументів або з декількома аргументами. Комбінація, що подана в дужках `(int... v)` неявно оголошується як масив типу `int[]`. Таким чином в тілі методу `vaTest()` доступ до масиву `v` буде забезпечуватись за допомогою синтаксису звичайного масиву. Далі розглянемо приклад застосування методу з передачею йому динамічної кількості аргументів:

```
public class VarArgs {
	static void vaTest(int... v) {
		System.out.println("Кількість аргументів: " + v.length);

		for (int x : v) // застосування циклу fof each
			System.out.println(x + " ");

		System.out.println();
	}

	public static void main(String[] args) {

		// існує декілька способів виклику методу vaTest
		// з аргументами динамічної довжини

		vaTest(10); // 1 аргумент
		vaTest(1, 2, 3); // 3 аргументи
		vaTest(); // без аргументів
	}
}
```

Результат роботи методу:

```
Кількість аргументів: 1
10

Кількість аргументів: 3
1
2
3

Кількість аргументів: 0
```

Слід відмітити дві важливі особливості наведеної в прикладі програми. По перше, які було зазначено раніше, в тілі методу `vaTest()` змінна `v` діє як масив (оскільки вона насправді є масивом). Синтаксична конструкція «. . .» вказує компілятору, що в даному методі буде використовуватись динамічна кількість аргументів та, що ці аргументи будуть зберігатись в масиві `v`.

По друге, метод `vaTest()` викликається в методі `main()` з різною кількістю аргументів, у тому числі і зовсім без них. Аргументи автоматично розміщуються в масиві та передаються змінній `v`. Якщо аргументи відсутні, до довжина масиву буде рівна 0.

Поряд з аргументами динамічної довжини методу можуть також передаватись звичайні аргументи. Проте при написанні такого методу слід врахувати, що параметр динамічної довжини має бути останнім серед усіх параметрів, оголошених в методі:

```
int doIt (int a, int b, double c, int ...vals)
```

В даному випадку, при виклику подібного методу, перші три аргументи будуть відповідати першим трьом параметрам. А всі решта аргументів будуть належати до масиву `vals`. Якщо недотримуватись правил подання динамічної кількості аргументів, то компілятор видаватиме помилку:

```
int doIt (int a, int b, int ...vals, double c)// помилка компіляції
```

Існує ще одне обмеження, яке слід враховувати: метод може містити лише один аргумент динамічної довжини (параметр з динамічною кількістю аргументів). До прикладу наведений нижче метод також виведе помилку компіляції:

```
int doIt (int a, int b, int ...vals, double ...c)// помилка компіляції
```

Спроба оголосити другий параметр з динамічною кількістю аргументів недоступна. Далі наведено приклад програмного коду, що приймає як звичайні аргументи, так і аргументи динамічної довжини:

```
public class VarArgs {
```

```
	static void vaTest(String msq, int... v) {
		System.out.print(msq + "Зміст: ");

		for (int x : v)
			System.out.print(x + " ");

		System.out.println();
	}

	public static void main(String[] args) {
		vaTest("Один аргумент: ", 10);
		vaTest("Три аргументи: ", 1, 2, 3);
		vaTest("Без аргументів ");
	}
}
```

Результат роботи:

```
Один аргумент: Зміст: 10
Три аргументи: Зміст: 1 2 3
Без аргументів Зміст:
```

Висновок: основний функціонал (логіка) бідь-якого додатку подається у методах, які використовуючи значення змінних екземплярів класу виконують основну роботу програми. Як вже відомо клас є оболонкою для написання програми, а методи це безпосередні «виконавці» цієї програми. Саме тому знання особливостей написання методів, їх структури та порядку застосування є одними із ключових моментів для розв'язання прикладних завдань з програмування на практиці.

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. 162-168, 182-187, 203-206, 207-208)** ***(наявне в віртуальному середовищі)***
3. Head First Java (изучаем Java): пер. с англ. / Kathy Sierra, Bert Bates. – Москва : «Эксмо», 2012. – 718 с. **(стор. 102-109)** ***(наявне в віртуальному середовищі)***

Додатково:

4. Програмування на Java (рос.) [Електронний ресурс]. – http://kostin.ws/java **(розділ 10) (*наявне в віртуальному середовищі)***
5. Освоюємо Java. Вікіпідручник [Електронний ресурс]. – Доступний з https://uk.wikibooks.org/wiki/Освоюємо_Java **(розділ 8)** ***(наявне в віртуальному середовищі)***
6. Java. ProgLang. Самоучитель (рос.) [Електронний ресурс]. – Доступний з http://proglang.su/java **(розділ 18) (*наявне в віртуальному середовищі)***