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 №6.1. ВВЕДЕННЯ В КЛАСИ

План:

1. Введення в класи
2. Конструктори

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. Введення в класи

Класи використовувались в прикладах програм ще з самого початку вивчення мови програмування Java. Але в розглянутих прикладах демонструвалась сама примітивна форма класу. Класи, у всіх раніше наведених прикладах, слугували лише контейнерами для методу `main()`, щоб ознайомити нас з синтаксисом та основами мови Java. Найбільш важлива особливість класу полягає в тому, що він являється шаблоном для створення об'єктів (екземплярів класу). Існує чудовий приклад цієї асоціації: клас – це як форма для випічки печива (форма одна, печива багато). Оскільки об'єкт є екземпляром класу то в Java поняття ***об'єкт*** та ***екземпляр класу*** є тотожними.

При визначенні класу оголошується його конкретна форма та сутність. Для цього необхідно вказати дані, що міститиме клас, а також код, який використовуватиме ці дані. Можуть зустрічатись класи, які містять лише код

або лише данні, проте більшість класів в реальних програмах містять обидва компоненти.

Якщо говорити термінологічною мовою то дані – це змінні екземпляра класу (поля класу), а код – це методи класу. ***Методи визначають те, що клас вміє, а змінні екземпляра класу (поля) – що він знає.***

Для оголошення класу слугує ключове слово **class**. Розглянемо загальну форму класу із усіма компонентами:

```
class Імя_класу {
      тип змінна_екземпляра1;
      тип змінна_екземпляра2;
      //...

      тип імя_методу1 (список_параметрів) {
            // тіло методу
      }
      тип імя_методу2 (список_параметрів) {
            // тіло методу
      }
      //...
}
```

В більшості класів, дії над змінними екземпляра і доступ до них реалізують методи, визначені в класі. Саме методи, як правило, визначають порядок використання змінних екземпляра класу. Змінні оголошенні в класі мають назву змінних екземпляра класу тому, що кожен екземпляр (об’єкт) містить власні копії оголошених змінних.

Для кращого розуміння розглянемо поняття класу на основі простого прикладу. Розглянемо код класу Box, який визначає три змінні екземпляра: `width`, `height`, `depth`. На даному етапі клас Box не містить жодних методів:

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

Клас визначає новий тип даних. В наведеному прикладі новий тип даних має назву Box. Це ім’я використовуватиметься для оголошення об’єктів типу Box. Слід зауважити, що застосування ключового слова **class** створює лише шаблон, проте не конкретний об’єкт. Щоб дійсно створити об’єкт (екземпляр) класу Box, потрібно застосувати такий оператор:

```
Box mybox = new Box() ; // створити об’єкт mybox класу Box
```

Після виконання подібного оператора об’єкт `mybox` набуде реальної сутності та являтиметься першим створеним екземпляром класу Box.

Кожен об’єкт, який створено на основі класу міститиме власну копію усіх змінних визначених в класі. В нашому прикладі, кожен об’єкт класу Box міститиме змінні екземпляра width, height, depth. Для доступу до змінної екземпляра застосовують оператор «.», який зв’язує ім’я об’єкту з ім’ям змінної екземпляра класу. Наприклад, для того, щоб присвоїти змінній width значення 100, потрібно виконати оператор:

```
mybox.width = 100;
```

Оператор «.» слугує доступом як до змінних екземпляра, так і до методів класу.

Поєднаємо усі розглянуті приклади у вигляді повноцінної програми з використанням класу Box:

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

// Клас із оголошенням об’єкту типу Box (екземпляра класу Box)
class BoxDemo {

      public static void main(String[] args) {

            Box mybox = new Box() ;

            double vol ; // змінна для запису результату розрахунку об’єму

            mybox.width = 10;
            mybox.height = 20;
            mybox.depth = 15;

            vol = mybox.width * mybox.height * mybox.depth; // розрахунок
об’єму

            System.out.println ( "Об'єм становить: " + vol ) ;
      }
}
```

Важливо! Після компіляції цієї програми буде створено два файли з вихідним кодом та розширенням *.class: один для класу Box, інший – BoxDemo. Два класи можуть розміщуватись як в одному так і в двох окремих файлах з розширенням *.java. На результат виконання програми це не впливатиме. Єдине, за умови рознесення класів різними файлами, для запуску програми необхідно запускати той файл, який містить main()-метод. А за умови розміщення класів в одному файлі, його слід називати ім’ям класу з main()-методом, тобто BoxDemo.java.

За умови наявності декількох екземплярів класу, коригування змінних одного екземпляра не впливатиме на змінні іншого екземпляра класу. Приклад:

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

class BoxDemo {

	public static void main(String[] args) {

		Box mybox1 = new Box() ;
		Box mybox2 = new Box() ;
		double vol ;
		// присвоєння значень змінним першого екземпляру
		mybox1.width = 10;
		mybox1.height = 20;
		mybox1.depth = 15;
		// присвоєння значень змінним другого екземпляру
		mybox2.width = 5;
		mybox2.height = 10;
		mybox2.depth = 7.5;

		vol = mybox1.width * mybox1.height * mybox1.depth;
		System.out.println ( "Об'єм 1 коробки становить: " + vol ) ;

		vol = mybox2.width * mybox2.height * mybox2.depth;
		System.out.println ( "Об'єм 2 коробки становить: " + vol ) ;
	}
}
```

Результат роботи програми становить:

```
Об'єм 1 коробки становить: 3000.0
Об'єм 2 коробки становить: 375.0
```

Як видно змінні одного екземпляру класу ізольовані від змінних іншого екземпляру.

Оголошення об'єктів та посилання на об'єкти. Як відмічено раніше, під час створення нового класу створюється новий тип даних (посилкового типу), який можна використовувати для оголошення об'єктів даного типу. Але створення об'єктів класу представляє собою двоетапний процес. Спочатку слід оголосити змінну типу класу, в наведеному прикладі це змінну типу `Box`. Ця змінна не визначає об'єкт, вона являється лише змінною, що може посилатись на об'єкт. Після створення змінної необхідно отримати конкретну фізичну копію об'єкта та присвоїти її оголошеній змінній. Це можливо виконати з допомогою оператора **`new`**. Цей оператор резервує пам'ять для об'єкта та повертає посилання на нього. Посилання – це адреса об'єкта в пам'яті. Отримане посилання зберігається в оголошеній змінній. Таким чином,

оперативна пам'ять під об’єкти в Java виділяється динамічно. Розглянемо описану процедуру детальніше.

Як нам вже відомо, стрічка наведена нижче слугує для оголошення об’єкту типу Box з ім’ям mybox:

```
Box mybox = new Box();
```

В наведеній стрічці об’єднуються два етапи щойно описаного процесу. Для того, щоб кожен етап став очевидним, наведену стрічку можна переписати наступним чином:

```
Box mybox; // оголошення змінної типу Box
mybox = new Box() ; // виділення пам’яті для об’єкта типу Box та присвоєння посилання на об’єкт (адреси в пам’яті) змінній mybox
```

В першій стрічці змінна mybox оголошується як шаблон посилання на об’єкт типу Box. На даний момент змінна mybox не має посилання на конкретний об’єкт. В наступній стрічці коду виділяється пам'ять для конкретного об’єкту, а змінній mybox присвоюється посилання на цей об’єкт. Після виконання другої стрічки коду, змінну mybox можна використовувати так, наче вона є об’єктом типу Box. Але насправді змінна mybox лише містить адресу комірки в пам’яті конкретного об’єкту типу Box.

Далі розглянемо процеси присвоювання змінним посилань на об’єкти. Для наочності розглянемо приклад:

```
Box b1 = new Box();
Box b2 = b1;
```

Після виконання представленого фрагменту коду, дві змінні b1 та b2 будуть мати посилання на один об’єкт. Присвоювання однієї змінної іншій не супроводжується виділенням пам’яті або копіюванням об’єкту. Таке присвоювання призведе лише до того, що змінна b2 матиме посилання на той самий об’єкт, що й змінна b1.

Крім того, що змінні b1 та b2 посилаються на один об’єкт, вони більше не зв’язані жодним чином. Наприклад, якщо в наведеному нижче прикладі присвоїти пусте значення (null) змінній b1, то в результаті лише втратиться зв’язок змінної з об’єктом, не завдаючи жодного впливу на сам об’єкт та змінну b2:

```
Box b1 = new Box();
Box b2 = b1;
b1 = null;
```

Підсумовуючи наведений матеріал розглянемо відмінність поняття класу від об’єкта. Клас створює новий тип даних (посилковий), який можливо використовувати для створення об’єктів. Це означає, що клас створює логічний

каркас, який визначає взаємозв'язок між його членами. При оголошенні об'єкта класу створюється екземпляр цього класу. Таким чином, клас – це логічна конструкція, а об'єкт володіє фізичною сутністю, тобто займає конкретну область оперативної пам'яті.

2. Конструктори

Ініціалізація усіх змінних класу при створенні його екземплярів може виявитись виснажливим та кропітким процесом. Ефективним способом ініціалізації змінних є присвоєння їм певних значень під час першого згадування про об'єкт (під час його створення). В зв'язку з тим, що необхідність в ініціалізації виникає часто, об'єктам в Java дозволено виконувати власну ініціалізацію при створенні. Автоматична ініціалізація реалізується з допомогою конструктора.

Конструктор ініціалізує змінні екземпляра клусу безпосередньо під час створення об'єкту.

Синтаксис конструктора аналогічний синтаксису методів. Ім'я конструктора співпадає з ім'ям класу. Якщо конструктор визначений в класі, то він автоматично викликатиметься при створенні кожного екземпляра після завершення виконання оператора **`new`**. На відміну від методів, конструктори не мають типу даних, що повертається, навіть типу **`void`**. Конструктор ініціює внутрішній стан об'єкта таким чином, щоб код, який створює екземпляр, з самого початку містив повністю ініціалізований та придатний до використання об'єкт.

Розглянутий раніше приклад класу `Box` можливо переробити із використанням конструктора. В першому прикладі розглянемо простий конструктор, який встановлює сталі значення для змінних екземпляра класу.

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

	// Створюємо конструктор для класу Box та встановлюємо сталі
	// значення для усіх параметрів
	Box() {
		width = 10;
		height = 10;
		depth = 10;
	}

	// Створюємо метод для визначення об'єму
	double volume (){
		return width*height*depth;
	}
}
```

```
public class BoxDemo {

	public static void main(String[] args) {
		Box mybox1 = new Box ();
		Box mybox2 = new Box ();

		// отримати обєм першої коробки
		System.out.println(mybox1.volume());

		// отримати обєм другої коробки
		System.out.println(mybox2.volume());
	}
}
```

Результатом роботи програми буде виведення у консоль:

```
Об'єм складає 1000.0
Об'єм складає 1000.0
```

В наведеному прикладі, за будь-якого виклику методу `volume` результат буде сталий. Адже конструктором класу `Box` визначені сталі значення усіх змінних. Проте цей приклад наведений лише для наочності того, що за умови застосування конструктора зникає необхідність ініціалізації (присвоєння значень) параметрів об'єкта після його створення в `main()`методі.

Повернемось до оператора створення екземпляра класу. Після ключового слова **`new`** та ім'я класу, необхідно вказати круглі дужки. Власне ця операція завжди викликає конструктор класу.

```
Box mybox1 = new Box ();
```

Оператор **`new`** `Box ()` викликає конструктор `Box ()`. Якщо конструктор класу не визначено явно, то Java створює для будь-якого класу конструктор за замовчуванням (пустий конструктор). Конструктор створюється автоматично для усіх класів, саме тому за відсутності його явного приведення усі змінні класів (поля) можна ініціалізувати та присвоювати їм значення. Конструктор за замовчування ініціалізує всі змінні екземпляра встановлюючи їм значення за замовчуванням. Всі значення за замовчуванням можуть бути нульовими (пустими), а саме для чисельних типів – 0, для логічних – false. Зазвичай використання конструкторів за замовчуванням є достатнім для простих класів, проте не для складних. Як тільки в класі буде визначено власний конструктор, конструктор за замовчуванням більше не використовуватиметься.

Більшість конструкторів в програмуванні не встановлюють жодних даних, вони лише виконують ініціалізацію об'єкта. Їх називають параметризованими.

В попередньому прикладі конструктор класу `Box()` ініціалізував об'єкт, проте користі від цього було мало, так як всі об'єкти отримували однаковий

розмір. Проте в Java передбачено спосіб створення об’єктів класу з різними розмірами. Для цього достатньо ввести в конструктор параметри (параметризовані конструктори). Наприклад, в наведеному нижче прикладі визначається параметризований конструктор, який визначає розміри за значеннями параметрів конструктора. Зверніть увагу на порядок створення екземпляра класу (об’єкту):

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

	// Створюємо параметризований конструтор
	Box(double w, double h, double d) {
		width = w;
		height = h;
		depth = d;
	}

	// Створюємо метод для визначення об'єму
	double volume (){
		return width*height*depth;
	}
}

public class BoxDemo {
	public static void main(String[] args) {
		Box mybox1 = new Box (10,20,15);
		Box mybox2 = new Box (3,6,9);
		// отримати обєм першої коробки
		System.out.println("Об'єм складає " + mybox1.volume());
		// отримати обєм другої коробки
		System.out.println("Об'єм складає " + mybox2.volume());
	}
}
```

Результат роботи програми:

```
Об'єм складає 3000.0
Об'єм складає 162.0
```

Як видно з наведеного прикладу ініціалізація параметрів класу відбувається під час створення екземпляру за допомогою оператора **new**:

```
Box mybox1 = new Box (10,20,15);
```

За такого варіанту копії змінних `width, height, depth` будуть містити значення 10, 20, та 15 відповідно.

Досконалості немає меж, тому йдемо далі. Як відомо в Java не допускається оголошення двох локальних змінних з однаковими іменами в тій

самій області. Тим не менше допускається існування локальних змінних імена яких співпадають з іменами змінних екземпляра класу. Проте коли ім'я локальної змінної співпадає з ім'ям змінної екземпляра класу, локальна змінна ***перекриває*** змінну екземпляра. Саме тому іменами `width, height та depth` в класі `Box` не були названі параметри конструктора `Box()`. В зворотному випадку ім'я `width`, наприклад, мало б значення формального параметру та скривало б змінну екземпляру `width`. І навіть якщо простіше обрати різні імена для локальних змінних і змінних екземпляра класу, існує й інший спосіб, більш цивілізований. Ключове слово **`this`** дозволяє посилатись безпосередньо на об'єкт, тому його можна використовувати для вирішення різних конфліктів, які можуть виникати між змінними екземпляру та локальними змінними. В якості прикладу розглянемо конструктор із застосуванням ключового слова **`this`**. В прикладі імена `width, height та depth` слугують для визначення параметрів, а ключове слово **`this`** – для звернення до змінних екземпляра за цими ж іменами.

```
Box(double width, double heighth, double depth) {
      this.width = width;
      this.height = heighth;
      this.depth = depth;
      }
```

В більшості професійно написаних програмах, розробники дотримуються правила щодо використання в конструкторі слова **`this`**. Таким чином в програмі використовуються одні і ті ж імена для оголошення одного і того ж параметру, що дозволяє внести більшу ясність в код програми.

Доречі, конструктор подібної архітектури також генерується IDE Eclipse, що підтверджує актуальність застосування ключового слова **`this`**. Для автоматичної генерації конструктора, після оголошення класу та змінних його екземпляру (полів класу), необхідно натиснути на комбінацію клавіш Shift+Alt+S. Після виконання цих операцій з'явиться контекстне меню на якому слід обрати Generate Constructor using Fields ...

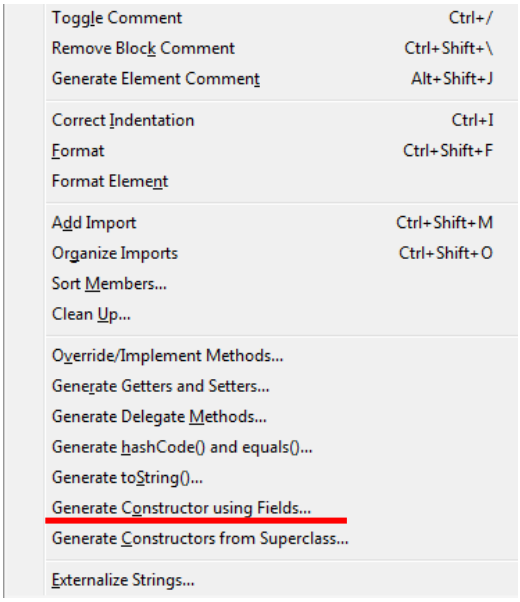

Рисунок 1 – Контекстне меню для автоматичного генерування конструктора

Після вибору відповідного меню середовище розробки виводить діалогове вікно з переліком змінних екземпляру класу, можливістю вибору модифікатора доступу, місця встановлення (після методів, спочатку тощо) та іншими опціями.

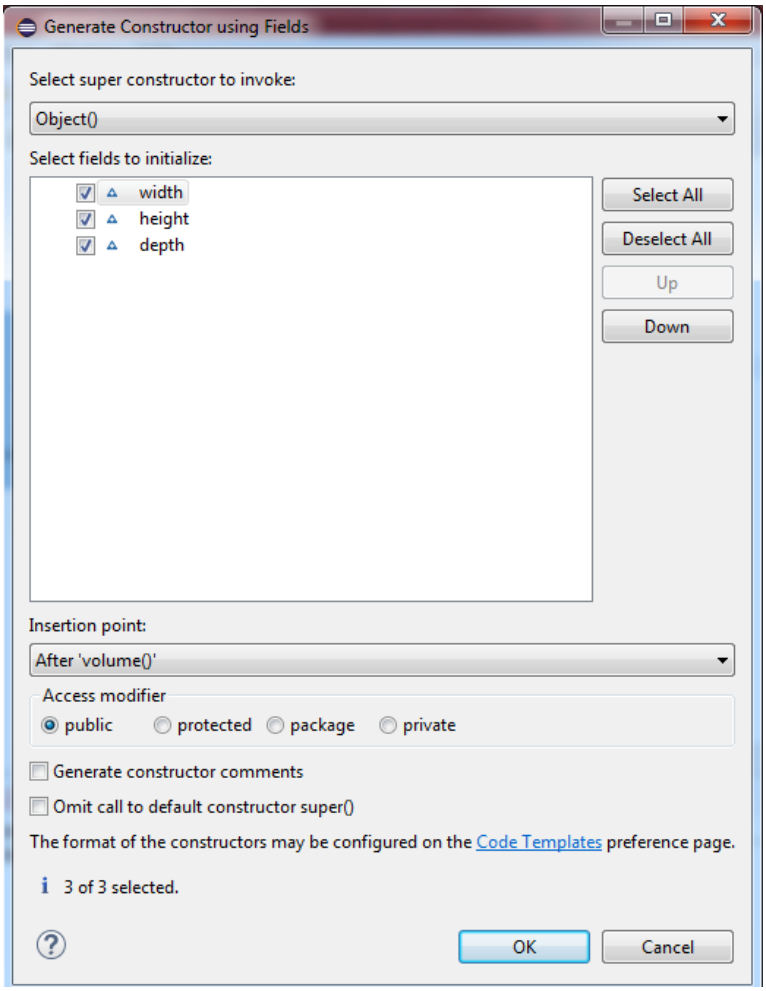

Рисунок 2 – Діалогове вікно з параметрами формування конструктора

Після виконання вказаних операцій клас Box матиме наступний вигляд:

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

	public Box(double width, double height, double depth) {
		this.width = width;
		this.height = height;
		this.depth = depth;
	}
		double volume() {
		return width * height * depth;
	}
}
```

В лекції слід згадати декілька слів про ***збірник сміття***. У випадку відсутності будь-яких посилань на об’єкт, в Java рахується, що цей об’єкт більше не потрібен та пам'ять, яку він займає можна звільнити. В інших мовах програмування таку операцію потрібно проводити власноруч, а в Java це відбувається автоматично.

Висновок: більшість сучасних додатків (прикладних, мобільних, системних) розробляються із застосуванням об’єктно-орієнтованого підходу, саме тому цей підхід вважається основним в прикладному програмуванні. Зважаючи на це зміст викладеної лекції є надзвичайно актуальним для кола початківців та є базовим для подальшого вивчення об’єктно-орієнтованого програмування.

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. 155-162, 168-172)** ***(наявне в віртуальному середовищі)***
3. Head First Java (изучаем Java): пер. с англ. / Kathy Sierra, Bert Bates. – Москва : «Эксмо», 2012. – 718 с. **(стор. 102-118)** ***(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 **(розділи 8, 9)** ***(наявне в віртуальному середовищі)***

6. Java. ProgLang. Самоучитель (рос.) [Електронний ресурс]. – Доступний з http://proglang.su/java **(розділи 5, 18, 21) (*наявне в віртуальному середовищі)***