Лабораторна робота №2
Тема: Основи роботи з класами та об'єктами. Інкапсуляція. Конструктор. Модифікатори доступу»

МЕТА ЛАБОРАТОРНОЇ РОБОТИ:

- **Ознайомитись з основами написання класів;**
- **Освоїти інкапсуляцію класів;**
- **Розібратися із використанням конструктора;**
- **Навчитися використовувати модифікатори доступу.**

ХОД РАБОТИ:

Об'єктно-орієнтоване програмування – методологія програмування, заснована на представленні програми як сукупності взаємодіючих об'єктів, кожен із яких є екземпляром певного класу.

ОБ’ЄКТ

Об'єкт – структура, яка поєднує дані та методи, які ці дані обробляють. Це дозволять розмежувати сферу застосування методів. Об'єкт – це будівельний блок об'єктно-орієнтованих програм. Об'єктно-орієнтована програма є, власне, набором об'єктів.

Об'єкт складається із трьох частин:

• ім'я об'єкта;

• стан (дані об'єкта, змінні стани). Стан об'єкта характеризується переліком всіх властивостей даного об'єкта та поточними значеннями кожного з цих властивостей;

• методи (операції).

Дані об'єктів. Дані, які у об'єкті, представляють його стан. У термінології ООП ці дані називаються атрибутами. Наприклад, атрибутами працівника може бути ім'я, прізвище, стать, дата народження, номер телефону тощо. У різних об'єктах атрибути мають різне значення.

Поведінка об'єктів. Поведінка об'єкта – те, що може зробити (у структурному програмуванні це реалізовувалося функціями, процедурами, підпрограмами).

Повідомлення – механізм комунікації між об'єктами. Наприклад, коли об'єкт А викликає метод об'єкта В, об'єкт А відправляє повідомлення об'єкту В. Відповідь

об'єкта В визначається його значенням, що повертається. Тільки «відкриті» методи можуть викликатись іншим об'єктом.

КЛАС

Кожен об'єкт визначається загальним шаблоном, який називається класом. У межах класу задається загальний шаблон, структура, основі якої потім створюються об'єкти. Дані, які стосуються класу, називаються полями класу, а програмний код їхньої обробки – методами класу. Поля та методи іноді називають загальним терміном – члени класу.

У класі описуються, якого типу дані відносяться до класу, а також те, які методи застосовуються до цих даних. Потім, у програмі з урахуванням тієї чи іншої класу створюється екземпляр класу (об'єкт), у якому вказуються конкретні значення полів і виконуються необхідні дії з них.

Відповідно до конвенцій коду Java:

• кожен клас повинен утримуватися у своєму окремому файлі з розширенням .java;

• назва файлу має співпадати з назвою класу;

• клас має бути іменником;

• ім'я класу має його описувати;

• ім'я класу починається з великої літери;

• якщо ім'я складається з кількох слів, кожне слово починається з великої літери.

Розглянемо різницю між об'єктом та класом на прикладі. Визначимо клас Cat та Dog. Визначення класу здійснюється через вказівку полів (даних) та методів класу. Для класу Cat як поля вкажемо name (кличку кота) і color (забарвлення). Для класу Dog задаємо поля name (кличка), color (забарвлення) та breed (порода).

Крім полів, визначимо методи цих класів. Методи – те, що може робити об'єкт класу (чи що можна робити з об'єктом). Коти нявкатимуть і ловитимуть мишей, а собаки гавкатимуть і вилятимуть хвостом.

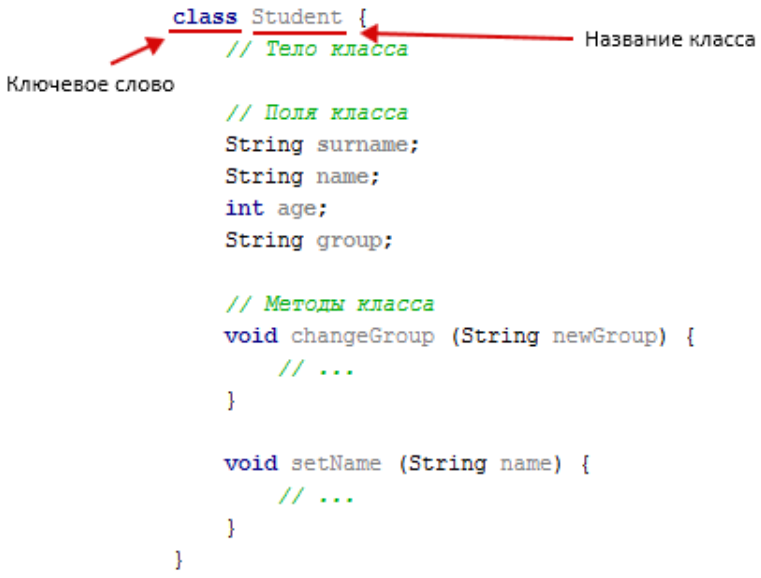
```java
class Dog {
    String name;     // Имя
    String color;    // Цвет
    String breed;    // Порода

    void bark() {
        // Лаять
    }

    void fawn() {
        // Вилять хвостом
    }
}
```

```java
class Cat {

    String name;     // Имя
    String color;    // Цвет

    void catchMouse() {
        // Ловить мышей
    }

    void mew() {
        // Мяукать
    }
}
```

Таким чином, ми визначили шаблони, на підставі яких згодом будуть створюватись екземпляри класів або об'єкти. Різниця між класом та об'єктом така сама, як між абстрактним поняттям та реальним об'єктом. Під час створення об'єкта класу задаються конкретні значення для полів. Коли ми говоримо про собаку чи кішку взагалі, як поняття, ми маємо на увазі домашніх тварин, які мають ім'я, забарвлення та інші характеристики. Це абстрактні поняття, які відповідають класу. А от якщо йдеться про конкретного Шарика чи Мурзика, то це вже об'єкти, екземпляри класу.

СИНТАКСИС КЛАСІВ

Розглянемо синтаксис опису класів Java. Опис класу починається із ключового слова class. Після цього слідує ім'я класу та у фігурних дужках тіло класу. Тіло класу складається з опису членів класу – полів та методів.



Для оголошення класу є ключове слово class. Спрощена форма визначення класу має вигляд:

```
class ім'я_класу {
     тип змінна_примірника_1;
     тип змінна_примірника_2;
     тип змінна_примірника_3;
...
     тип змінна_примірника_N;

     тип имя_метода_1(список параметрів) {...}
     тип имя_метода_2(список параметрів) {...}
...
     тип имя_метода_N(список параметрів) {...}
}
```

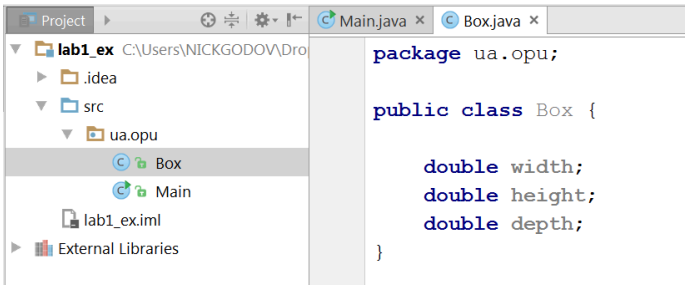
Дані, або змінні, визначені у класі, називаються змінними примірниками. Код міститься у тілі методів. Змінні екземпляри та методи називаються членами класу. У більшості класів дії над змінними та доступ до них здійснюють методи цього класу. Отже, методи визначають порядок використання даних класу.

Змінні, визначені у класі, називаються змінними екземпляра, оскільки кожен екземпляр класу (тобто кожен об'єкт класу) містить власні копії цих змінних. Таким чином, дані одного об'єкта відокремлені та відрізняються від даних іншого об'єкта.

Важлива особливість класу у тому, що він визначає новий тип даних. Весь код, написаний Java, знаходиться в класах. Стандартна бібліотека Java містить тисячі класів, призначених для вирішення різних завдань, наприклад, для побудови графічного інтерфейсу, календарів, роботи з мережею і т.д..

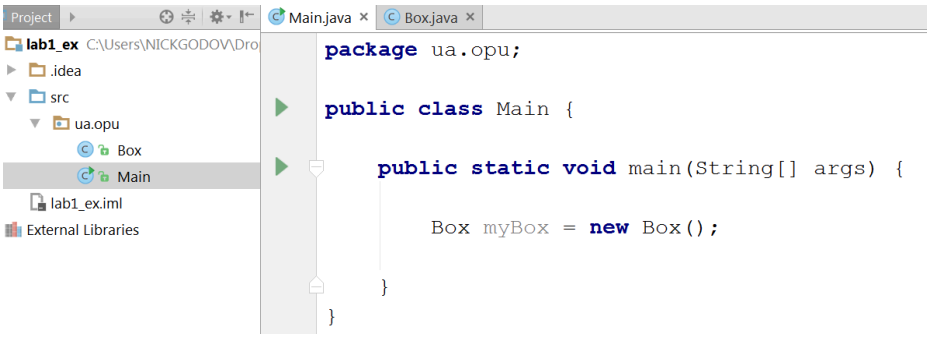
ПРИКЛАД КЛАСУ

Опишемо простий клас, який називатиметься Box. Визначимо три змінні: довжина, ширина і висота і вважатимемо, що вони можуть набувати тільки цілих (в Java рекомендується використовувати тип double замість float, тому немає сенсу економити 4 байти).



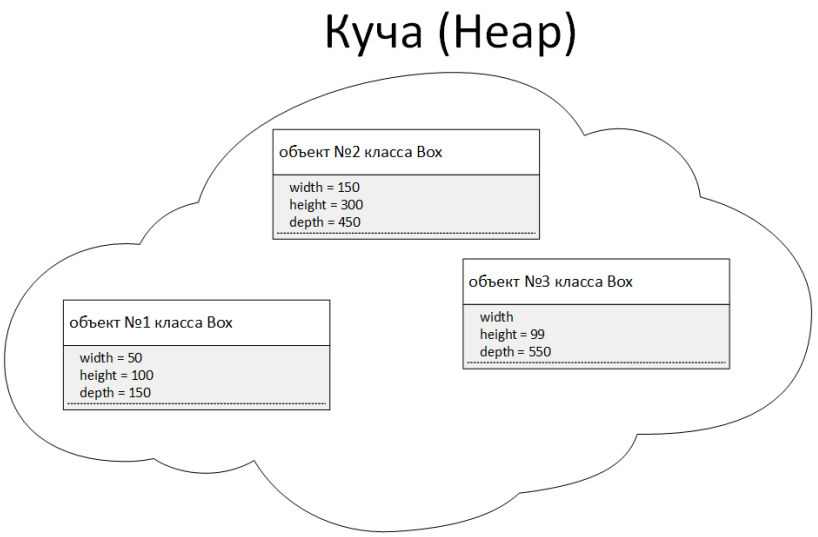
Як ми говорили, клас визначає тип даних. У разі новий тип даних називається Box. Це ім'я буде використано для оголошення типу Box. Не забуватимемо, що оголошення class створює тільки шаблон, але не конкретний об'єкт. Таким чином, поки що у нас є тільки шаблон, і немає об'єктів типу Box.

Щоб створити об'єкт класу Box, скористайтеся оператором new. Оператор new створює екземпляр зазначеного класу та повертає посилання на створений об'єкт.



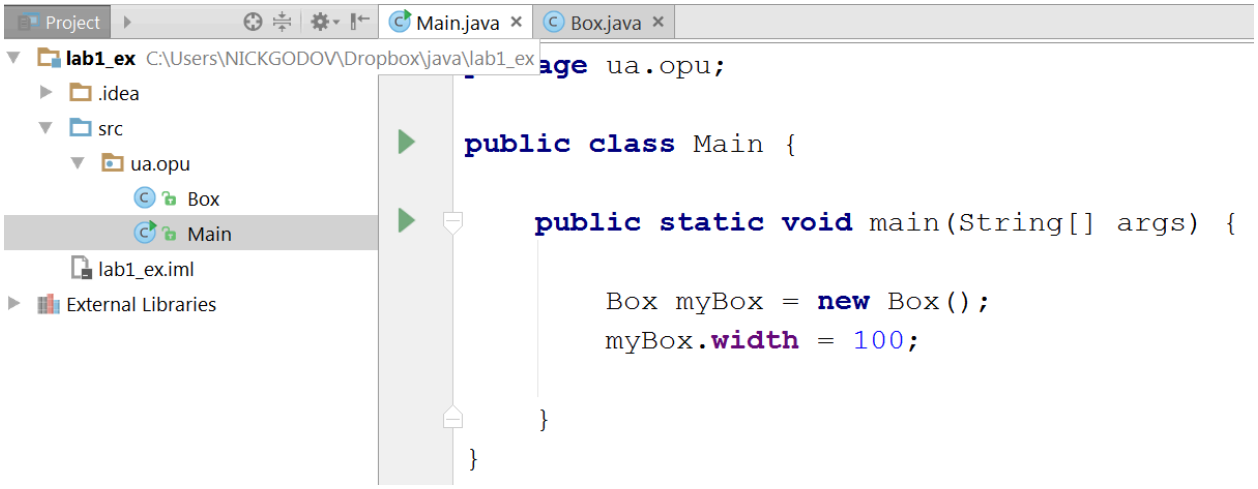
Після виконання цього оператора об'єкт myBox стане екземпляром класу Box.

Нагадаємо, що кожен об'єкт містить власну копію змінної екземпляра, яка визначена у класі. Таким чином, кожен об'єкт класу Box міститиме власні копії змінних примірників width, height, depth.



Зміни до змінних примірників одного об'єкта не впливають на змінні примірники іншого об'єкта.

Для доступу до цих змінних є операція-точка (.). Наприклад, щоб присвоїти змінній width об'єкту myBox значення 100, потрібно виконати наступний код:



Цей оператор наказує компілятору, що копії змінної width, яка зберігається в об'єкті myBox, потрібно присвоїти значення 100. За допомогою операції-точки можна отримати доступ до змінних екземпляра та методів у межах об'єкта.

ПРИКЛАД НАПИСАННЯ МЕТОДІВ КЛАСУ

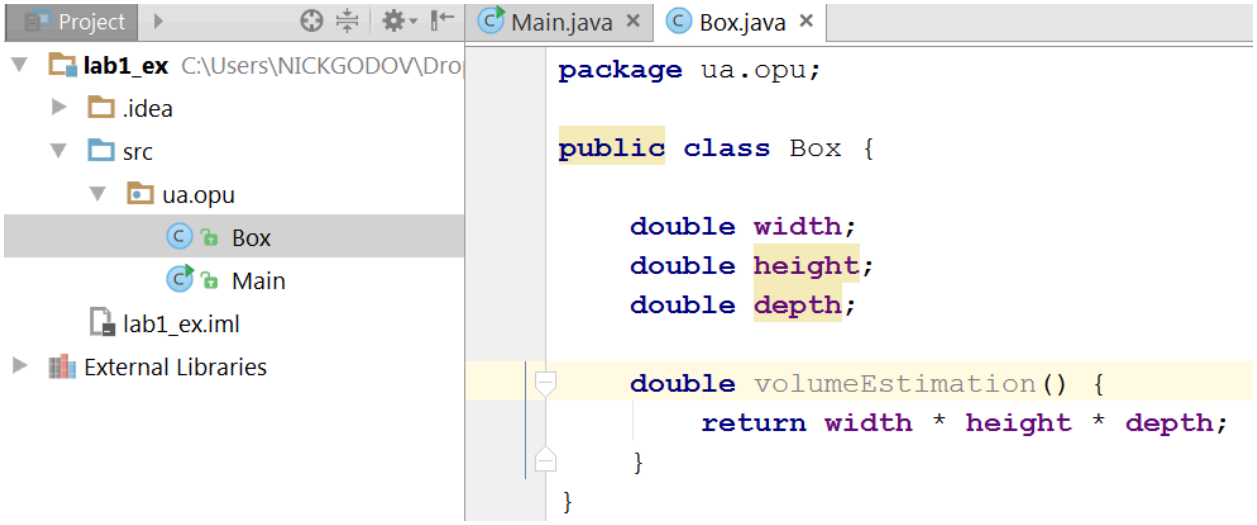
Загальна форма оголошення методу виглядає так:

```
тип имя_метода(список параметрів) {
      //тіло_метода
}
```

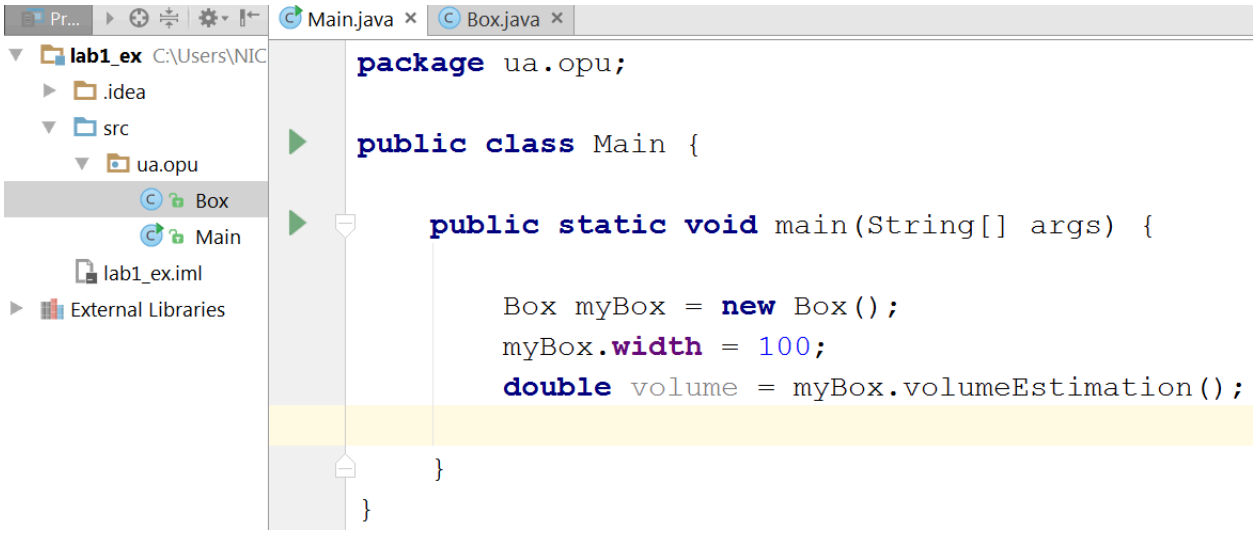
де тип позначає конкретний тип даних, який повертається методом. Він може бути будь-яким типом даних, у тому числі типом створеного класу.

Методи класу визначають інтерфейс класів. Це дозволяє тому, хто реалізує клас, приховувати внутрішню структуру класу та надавати назовні деякий набір методів.

Повернемося до класу Box та додамо метод для обчислення об'єму коробки:



Викличемо цей метод у об'єкта myBox, який ми створили раніше.



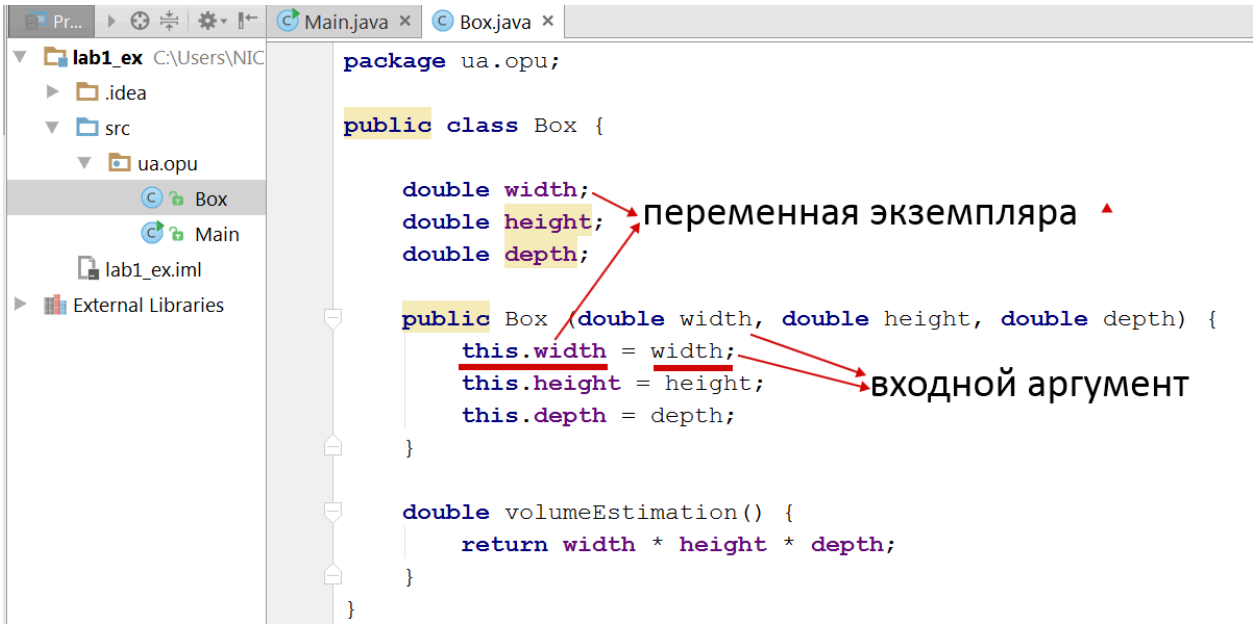
Зверніть увагу, що метод volumeEstimation() був викликаний по відношенню до об'єкта myBox. Це означає, що цей метод бере змінні width, height і depth об'єкта, у якого цей метод був викликаний.

Також слід звернути увагу, що коли ми описуємо метод volumeEstimation(), посилання на змінні екземпляра width, height і depth робиться безпосередньо без вказівки перед ними імені об'єкта або операції-точки. Коли методі використовується змінна екземпляра, визначена у його класі, це робиться безпосередньо, без зазначення явної посилання об'єкт і застосування операції-точки. Це означає, що змінні екземпляри width, height і depth неявно посилаються на копії цих змінних, що зберігаються в об'єкті, що викликає метод volumeEstimation().

КОНСТРУКТОР

Ініціалізація всіх змінних класу за кожного створення об'єкта – заняття досить стомлююче. У зв'язку з цим, Java дозволяється виконувати власну ініціалізацію при створенні об'єктів. Така ініціалізація здійснюється з допомогою конструктора.

Конструктор – це спеціальний метод, який викликається під час створення нового об'єкта. Синтаксис конструктора відрізняється від синтаксису звичайного методу. Його ім'я збігається з ім'ям класу, в якому він знаходиться, і він не має типу, що повертається.



КЛЮЧОВЕ СЛОВО THIS

Уявимо, що у нас є два об'єкти одного класу і для цих двох об'єктів викликається той самий метод:

```
Item apple = new Item("Яблоко");
Item pear = new Item("Груша");

Box box_1 = new Box();
Box box_2 = new Box();

box_1.setContents(apple);
box_2.setContents(pear);
```

Якщо існує один метод setContents(), як метод дізнається, якого об'єкта він викликається – для box_1 чи box_2?

Виявляється, при виклику методу setContents() (як і при викликі будь-якого іншого методу) передається прихований перший аргумент - посилання на об'єкт, що використовується. Таким чином, виклики методів насправді виглядають так:

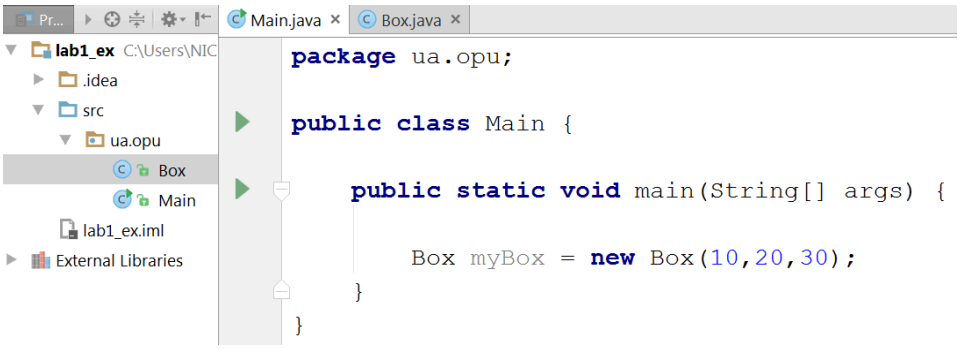
```
Item apple = new Item("Яблоко");
Item pear = new Item("Груша");

Box box_1 = new Box();
Box box_2 = new Box();

box_1.setContents(box_1, apple);
box_2.setContents(box_2, pear);
```

Іноді під час виконання методу необхідно отримати посилання на об'єкт, для якого був викликаний метод. Оскільки посилання на нього передається потай, ідентифікатора для неї немає. Але для цього існує спеціальне ключове слово – це. Ключове слово це надає посилання на об'єкт, для якого був викликаний метод. Поводитися з нею можна як і з будь-яким іншим посиланням на об'єкт. Для виклику методу класу з іншого методу цього ж класу, використовувати ключове слово this не потрібно.

```
public class Box {

    private Item contents;

    public void setContents(Item contents) {
        this.contents = contents;

        // Такой вараинт допустим, но смысла в нем нет
        this.somePrivateMethod();

        // Такой вариант более предпочтительный,
        somePrivateMethod();
    }

    private void somePrivateMethod() {
        // ... какой-то код ...
    }
}
```

У нашому випадку, імена аргументів конструктора збігаються з іменами змінних екземпляра, і, щоб звернутися саме до екземплярів, ми використовуємо ключове слово this і за допомогою оператора-точки звертаємось до змінної екземпляра та присвоюємо їй нове значення.

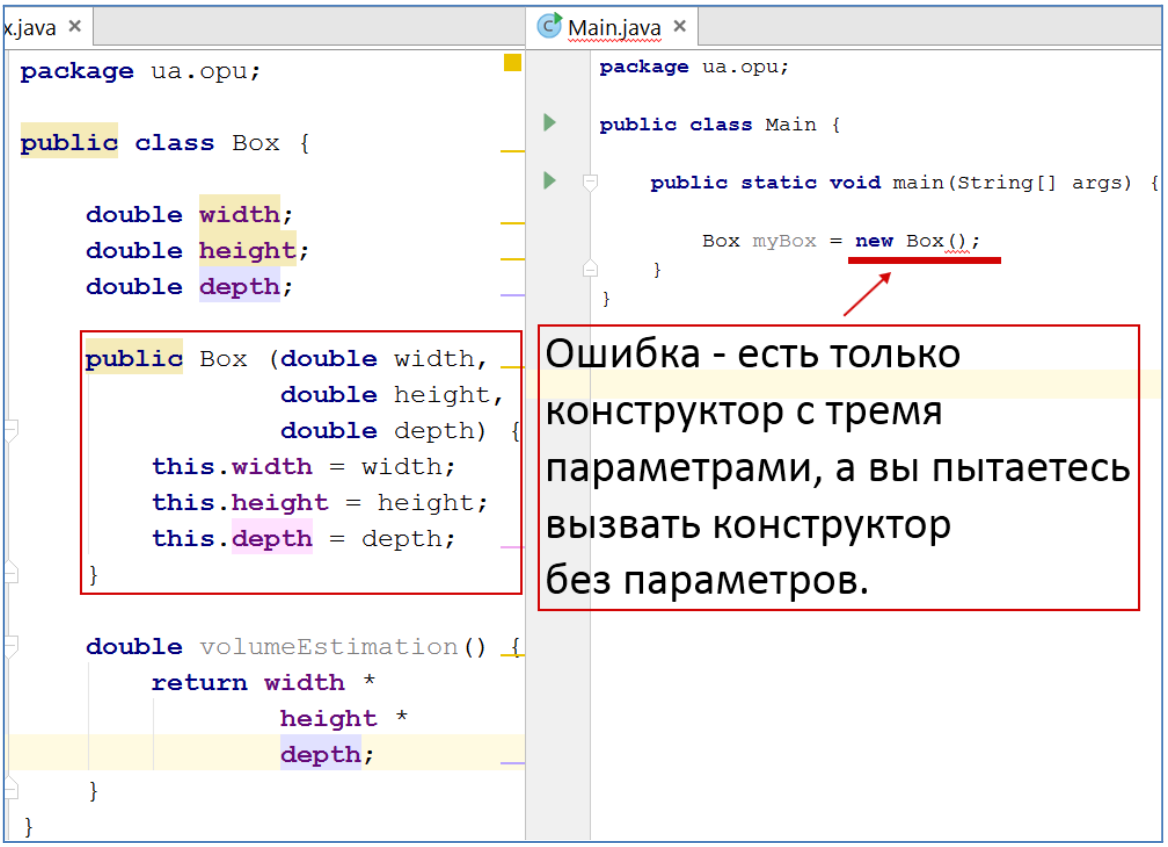
Тепер скористаємося конструктором під час створення об'єкта.



Тепер нам має бути зрозуміло, чому при створенні нового об'єкта після імені класу потрібно вказувати круглі дужки. Насправді оператор new викликає конструктор класу. Таким чином, у рядку коду

```
Box myBox = new Box();
```

Оператор new викликає конструктор Box(). Але ми раніше не створювали цей конструктор, чому компілятор не видав помилку, коли ми запускали програму? Якщо в класі не визначите конструктор, Java створює в класі конструктор за замовчуванням. Конструктор за замовчуванням ініціалізує всі змінні значеннями за умовчанням (null для змінних, що посилаються на об'єкти, false для змінних типу boolean і 0 для решти всіх примітивних типів). Якщо ж ви визначите в класі хоча б один конструктор, то за замовчуванням конструктор створений не буде. Саме тому наступний код видасть помилку.



ІНКАПСУЛЯЦІЯ І ОБМЕЖЕННЯ ДОСТУПУ

Інкапсуляція – один із основоположних принципів ОВП. Інкапсуляція – це одна з причин, чому широко використовується ООП.

Інкапсуляцію можна вважати захисною оболонкою, яка оберігає код і дані від довільного доступу з іншого коду, що знаходиться зовні оболонки. Доступ до коду та даних, що знаходяться всередині оболонки, суворо контролюється ретельно певним інтерфейсом (набором загальнодоступних, публічних методів).

У будь-якому класі є дві частини: інтерфейс і реалізація.

Інтерфейс відображає зовнішню поведінку об'єктів цього класу. Внутрішня реалізація визначає уявлення та механізми досягнення бажаної поведінки об'єкта.

В інтерфейсі зібрано все, що стосується взаємодії даного об'єкта з іншими об'єктами, а реалізація приховує від інших об'єктів усі деталі, що не стосуються процесу взаємодії об'єктів.

Інкапсуляція - це процес відокремлення один від одного елементів об'єкта, що визначають його пристрій та поведінки; інкапсуляція служить у тому, щоб ізолювати контрактні зобов'язання від реалізації.

Інкапсуляція дозволяє локалізувати частини реалізації системи, які можуть зазнати змін. У міру розвитку програми, розробники можуть ухвалити рішення змінити внутрішній пристрій тих чи інших об'єктів з метою покращення продуктивності або економії пам'яті. Але інтерфейс буде незайманим і дозволить іншим об'єктам у такий же спосіб взаємодіяти з цим об'єктом. (Приклад автомобіля – педалі, кермо, панель приладів і внутрішня начинка).

Інкапсуляція Java реалізована за допомогою використання модифікаторів доступу.

Мова Java надає кілька рівнів захисту, які дозволяють настроювати область видимості даних та методів. Java має чотири категорії видимості елементів класу:

• private – члени класу доступні лише членам цього класу;

• за замовчуванням (package-private) – члени класу доступні класам, які знаходяться у цьому пакеті;

• protected – члени класу доступні класам, які у тому пакеті, і підкласам – інших пакетах;

• public – члени класу доступні для всіх класів у цьому та інших пакетах.

Член класу (змінна, конструктор, методи), оголошений public, доступний з будь-якого методу поза класом.

Все, що оголошено private, доступне тільки конструкторам і методам усередині класу і ніде більше. Вони виконують службову чи допоміжну роль межах класу та його функціональність не призначена для зовнішнього користування. Закриття (private) полів забезпечує інкапсуляцію.

ПРИКЛАД ВИКОРИСТАННЯ ІНКАПСУЛЯЦІЇ

Уявимо, що нам необхідно створити клас «Кошик» (Cart), який зберігає набір об'єктів класу «Товар» (Item).

Які методи «Кошик» має надавати для зовнішнього використання. Це можуть бути, наприклад, методи «Додати товар», «Прибрати останній доданий товар», «Підрахунок суми цін товарів у кошику», «Підвищення цін у кошику на N відсотків» та «Зниження цін у кошику на N відсотків».

Назва методу	Опис
`public Cart(int capacity);`	Конструктор з 1 параметром – максимальною кількістю товарів у кошику
`public boolean addItem(Item item);`	Додавання товару до кошика. Повертає успішність операції.

`public Item deleteLastAddedItem();`	Видалення останнього доданого товару до кошика. Повертає віддалений товар.
`public double calculateItemPrices();`	Підрахунок суми цін усіх товарів у кошику.
`public void raiseItemPrices(double percent);`	Підняти ціни товарів у кошику певний відсоток (значення відсотка передається як аргумент методу).
`public void cutItemPrices(double percent);`	Знизити ціни товарів у кошику певний відсоток (значення відсотка передається як аргумент методу).

Як ви можете помітити, це public-методи, а значить, їх можна викликати через оператор-точку, маючи посилання на об'єкт.

Cart.java
```
package ua.opu;

public class Cart {

    public boolean addItem (Item item) {
        // ... Выполняется какой-то код ...
        return true;
    }
}
```
Main.java
```
public class Main {

    public static void main(String[] args) {
        Cart cart = new Cart();
        // public-метод можно вызвать через оператор-точку
        cart.addItem(new Item());
    }
}
```

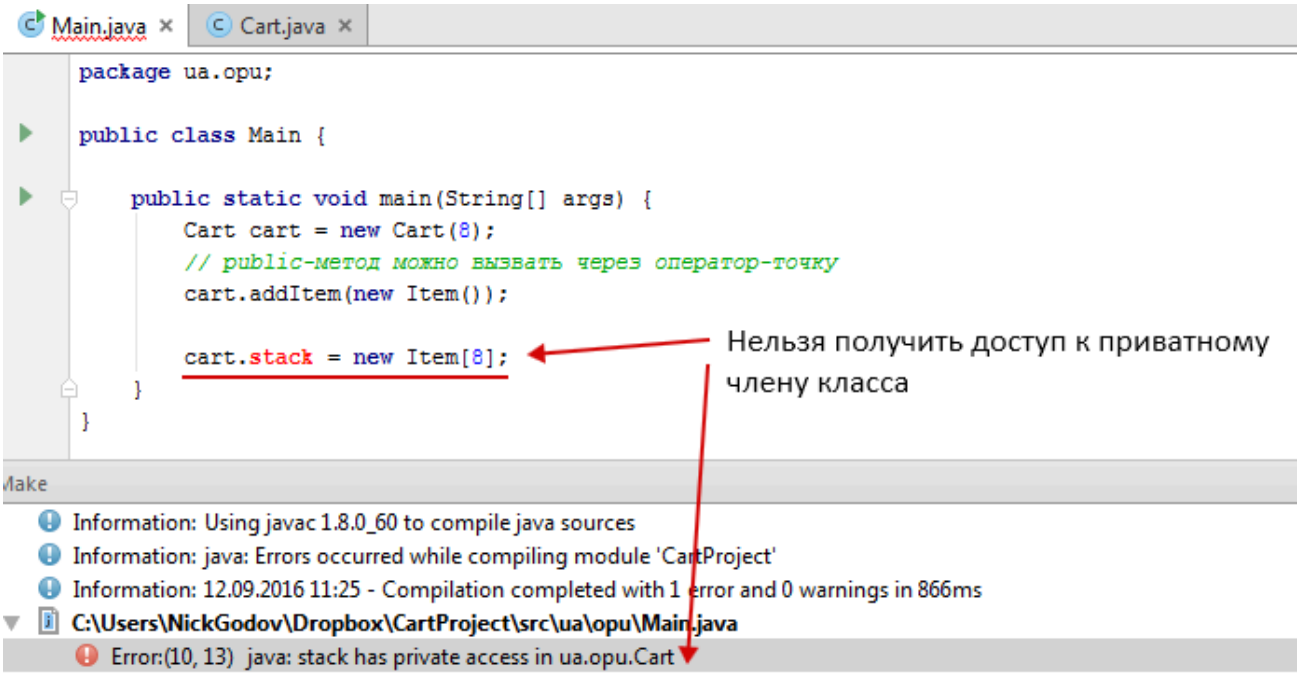
Перелік цих public-методів і складає інтерфейс класу – тобто за допомогою цих методів об'єкт класу взаємодіятиме із зовнішнім світом. Ці методи мають цілком чітко визначені вхідні аргументи і можуть повертати значення чітко визначених типів, і ніяк інакше. За аналогією з цим, поворот коліс автомобіля здійснюється чітко певним чином – поворотом керма, і бензин треба заливати у чітко певний отвір кришки бензобака, а не ще якось.

Те – як буде реалізовано зберігання товарів у кошику – це внутрішня логіка класу і вона має бути доступна зовнішньому світу, вона має бути прихована від зовнішнього втручання. Інші класи, які будуть використовувати об'єкти класу Cart не повинні знати і не повинні мати доступ до того - як там «всередині» реалізовано зберігання товарів, підрахунок цін і зміна ціни на певний відсоток і т.д., вони можуть тільки використати надані їм public-методи.

Давайте реалізуємо «Кошик» за допомогою структури «стек», яка, у свою чергу, реалізована звичайним масивом.

```
public class Cart {

    private Item[] stack; // масив для реалізації стека
    private int topIndex; // покажчик на вершину стека

    // При створенні кошика ми повинні
    // вказати максимальну кількість елементів
    // в кошику
    public Cart(int capacity) {
        stack = new Item [capacity];
        topIndex = -1;
    }

    // Додавання нового товару до кошика
    public boolean addItem(Item item) {
        return push(item);
    }

    // Приватний метод, який реалізує додавання до стек
    private boolean push (Item item) {
        // Додаємо товар у стек
        return true; // або false якщо не стек переповнений
    }

    // Видалення останнього доданого товару в кошик
    public Item deleteLastAddedItem() {
        return pop();
    }

    // Приватний метод, який реалізує витяг зі стека
    private Item pop() {
        return new Item(); // Витягнутий із стеку товар
    }
}
```

Як ми бачимо, масив з товарами, вказати на вершину стек оголошені як приватні члени класу. Це означає, що ми не можемо отримати до них доступ ззовні – вони доступні лише всередині класу.



Програміста, який буде використовувати клас Cart, не повинна хвилювати ситуація з переповненням стека, зі спробою витягти елемент з порожнього стека, він не повинен стежити за вказівником на вершину стека, він навіть не повинен знати, що це стек. Для нього об'єкт класу Cart – це певний об'єкт, який надає «послугу» у вигляді кошика товарів і з цим кошиком можна працювати за допомогою певних public-методів.

Надалі ми можемо переробити клас Cart та змінити внутрішню реалізацію. Ми можемо використовувати структуру черги, ми можемо використовувати колекції, ми можемо інакше реалізувати операції додавання та видалення елемента в стеку, але якщо ми збережемо інтерфейс класу незмінним, то для зовнішнього світу ці зміни внутрішньої логіки не будуть важливими і якщо ми змінимо внутрішню логіку одного невеликого ділянки програми, то вся решта програми працюватиме так само.

ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ:

1. Створіть клас "Товар" (Item).
А) Клас «Товар» має містити такі поля: найменування (рядок), ціна (float).
Б) Клас «Товар» повинен містити конструктор, який приймає два параметри: найменування та початкову ціну товару.
В) Клас «Товар» повинен мати такі публічні методи: підвищення ціни на певний відсоток (значення відсотка типу float передається як аргумент методу); зниження ціни певний відсоток (значення відсотка типу float передається як аргумент методу).
Г) У класі має бути реалізована перевірка ціни на негативне значення. Якщо конструкторі передається негативне значення ціни, чи результаті зниження ціни на >100% ціна стає негативною, вона має бути примусово встановлено 0.

2. Створіть клас «Кошик» (Cart).
А) «Кошик» має реалізовувати структуру даних стек, у якому містяться об'єкти класу «товар». Вважатимемо, що ми додаємо завжди по 1 одиниці товару. Стек «всередині» реалізується традиційним масивом об'єктів Item. Клас повинен містити перевірки, пов'язані з роботою стека - переповнення стека, спроба витягти елемент із порожнього стека і т.д.
Б) «Кошик» повинен містити конструктор з 1 параметром – максимальною кількістю елементів у стеку. У цьому архітекторі відбувалася ініціалізація масиву, який реалізує стек.
В) «Кошик» повинен містити такі публічні методи: додавання товару, видалення товару, підрахунок суми цін товарів у кошику (пройтися за елементами масиву та скласти всі значення цін), підвищення та зниження цін усіх товарів у стеку (два окремі методи, значення ціни передається як параметр методу, необхідно пройтись по всіх елементах масиву та передати відповідне повідомлення об'єктам.

3. У методі Main необхідно створити об'єкт класу «Кошик» з деякою максимальною кількістю елементів у стеку.
А) Заповнити кошик об'єктами класу Item (5-6 об'єктів буде достатньо);
Б) Вивести суму цін товарів усередині кошика;
В) Підняти ціни в кошику на 15%, вивести змінену суму цін на консоль.
Г) Знизити ціни в кошику на 30%, вивести змінену суму цін на консоль.

4 . Перепишіть клас Cart, використовуючи замість стека чергу (або колекцію, якщо ви знаєте, що таке) без зміни інтерфейсу класу. Перевірте роботу коду.

Контрольні питання:

1. Що таке програма з погляду об'єктно-орієнтованої парадигми?
2. Що таке об'єкт?
3. З яких частин складається об'єкт?
4. Що таке клас?
5. З яких частин складається клас?
6. Як створити об'єкт якогось класу?
7. Що таке конструктор? Що таке конструктор за замовчуванням?
8. Що таке інкапсуляція? Навіщо вона потрібна? Як реалізовано інкапсуляцію в мові Java?