МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

Одеський національний політехнічний університет

Інститут Комп’ютерних Систем

Кафедра Інформаційних Систем та Технологій

Протокол лабораторної роботи №2

з дисципліни об’єктно-орієнтоване програмування

на тему: « РАЗРАБОТКА КОНСОЛЬНЫХ ПРИЛОЖЕНИЙ »

Виконав студент групи

АД-201

Стицковський Н.Ю.

Прийняв

Рудніченко Н.Д.

Одеса, 2021

**Содержание**

Введение ……. 1

Теоретическая часть ….. 2

Практическая часть ……. 3

Вывод …….. 4

Литература …….. 5

ВВЕДЕНИЕ

ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:

* Ознакомиться с основами написания классов;
* Освоить инкапсуляцию классов;
* Разобраться с использованием конструктора;
* Научиться использовать модификаторы доступа.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

**ОБЪЕКТ**

**Объект** – структура, которая объединяет данные и методы, которые эти данные обрабатывают. Это позволят разграничить область применения методов. **Объект** – это строительный блок объектно-ориентированных программ. Объектно-ориентированная программа является, по сути, набором объектов.

Объект состоит из трех частей:

* имя объекта;
* состояние (данные объекта, переменные состояния). Состояние объекта характеризуется перечнем всех свойств данного объекта и текущими значениями каждого из этих свойств;
* методы (операции).

**Данные объектов.** Данные, содержащиеся в объекте, представляют его состояние. В терминологии ООП, эти данные называются атрибутами. Например, атрибутами работника могут быть имя, фамилия, пол, дата рождения, номер телефона и т.д. В разных объектах атрибуты имеют разное значение.

**Поведение объектов.** Поведение объекта – то, что он может сделать (в структурном программировании это реализовывалось функциями, процедурами, подпрограммами).

**Сообщения** – механизм коммуникации между объектами. Например, когда объект А вызывает метод объекта B, объект A отправляет сообщение объекту B. Ответ объекта B определяется его возвращаемым значением. Только «открытые» методы могут вызываться другим объектом.

**КЛАСС**

Каждый объект определяется общим шаблоном, который называется классом. В рамках класса задается общий шаблон, структура, на основе которой затем создаются объекты. Данные, относящиеся к классу, называются полями класса, а программный код для их обработки – методами класса. Поля и методы иногда называют общим термином – члены класса.

В классе описываются, какого типа данные относятся к классу, а также то, какие методы применяются к этим данным. Затем, в программе на основе того или иного класса создается экземпляр класса (объект), в котором указываются конкретные значения полей и выполняются необходимые действия над ними.

Согласно конвенциям кода Java:

• каждый класс должен содержаться в своем отдельном файле с расширением .java;

• название файла должно совпадать с названием класса;

• класс должен быть именем существительным;

• имя класса должно его описывать;

• имя класса начинается с большой буквы;

• если имя состоит из нескольких слов, то каждое слово начинается с большой буквы.

Рассмотрим разницу между объектом и классом на примере. Определим класс Cat и Dog. Определение класса производится через указание полей (данных) и методов класса. Для класса Cat в качестве полей укажем name (кличку кота) и color (окрас). Для класса Dog задаем поля name (кличка), color (окрас) и breed (порода).

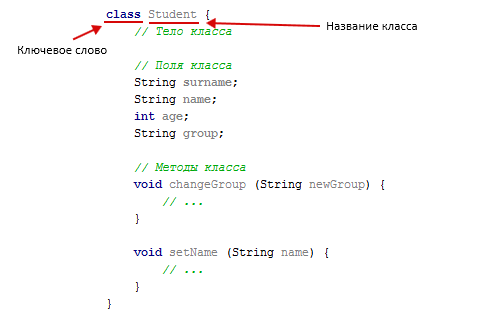
Помимо полей, определим методы для этих классов. Методы – это то, что может делать объект класса (или что можно делать с объектом). Коты будут мяукать, и ловить мышей, а собаки лаять и вилять хвостом.

|  |  |
| --- | --- |
|  |  |

Таким образом, мы определили шаблоны, на основании которых впоследствии будут создаваться экземпляры классов или объекты. Разница между классом и объектом такая же, как между абстрактным понятием и реальным объектом. При создании объекта класса задаются конкретные значения для полей. Когда мы говорим о собаке или кошке вообще, как понятии, мы имеем в виду домашних животных, у которых есть имя, окрас и прочие характеристики. Это абстрактные понятия, которые соответствуют классу. А вот если речь идет о конкретном Шарике или Мурзике, то это уже объекты, экземпляры класса.

**СИНТАКСИС КЛАССОВ**

Рассмотрим синтаксис описания классов в Java. Описание класса начинается с ключевого слова class. После этого следует имя класса и в фигурных скобках тело класса. Тело класса состоит из описания членов класса – полей и методов.



Для объявления класса служит ключевое слово ***class***. Упрощенная форма определения класса имеет вид:

|  |
| --- |
| class имя\_класса {  тип переменная\_экземпляра\_1;  тип переменная\_экземпляра\_2;  тип переменная\_экземпляра\_3;  ...  тип переменная\_экземпляра\_N;  тип имя\_метода\_1(список параметров) {...}  тип имя\_метода\_2(список параметров) {...}  ...  тип имя\_метода\_N(список параметров) {...}  } |

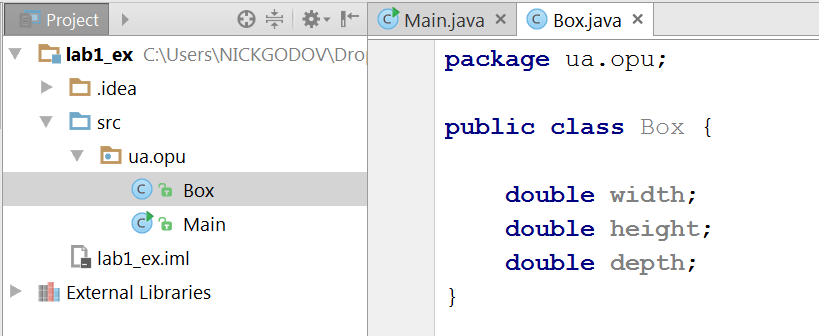
Данные, или переменные, определенные в классе, называются *переменными экземпляра*. Код содержится в теле методов. Переменные экземпляра и методы называются членами класса. В большинстве классов действия над переменными и доступ к ним осуществляют методы этого класса. Таким образом, методы определяют порядок использования данных класса.

Переменные, определенные в классе, называются переменными экземпляра, поскольку каждый экземпляр класса (то есть, каждый объект класса) содержит собственные копии этих переменных. Таким образом, данные одного объекта отделены и отличаются от данных другого объекта.

Важная особенность класса состоит в том, что он определяет новый тип данных. Весь код, написанный на Java, находится в классах. Стандартная библиотека Java содержит тысячи классов, предназначенных для решения самых разных задач, например, для построения графического интерфейса, календарей, работы с сетью и т.д.

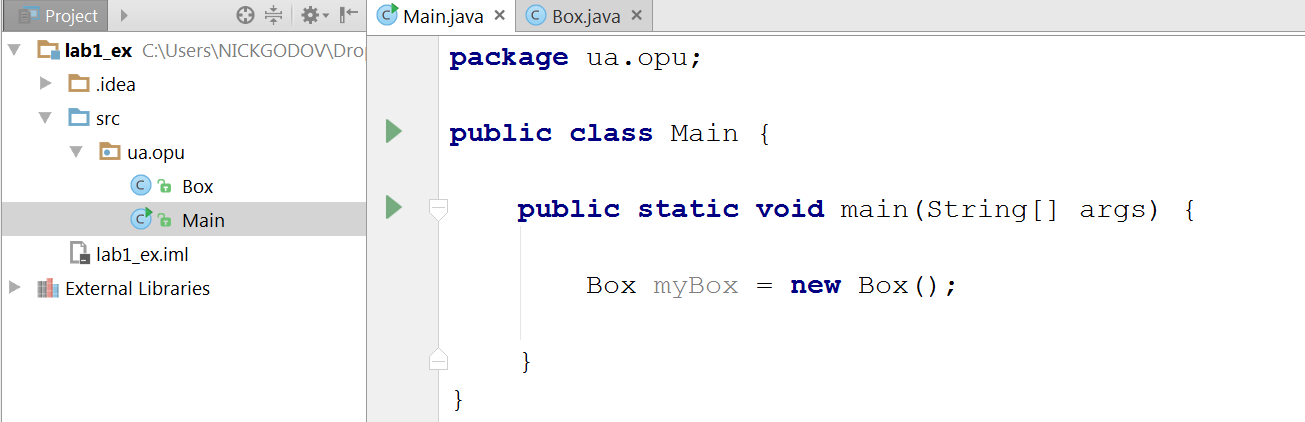
**ПРИМЕР КЛАССА**

Опишем простой класс, который будет называться Box. Определим три переменные: длина, ширина и высота и будем считать, что они могут принимать только целочисленные значения (в Java рекомендуется использовать тип double вместо float, т.к. нет смысла экономить 4 байта).



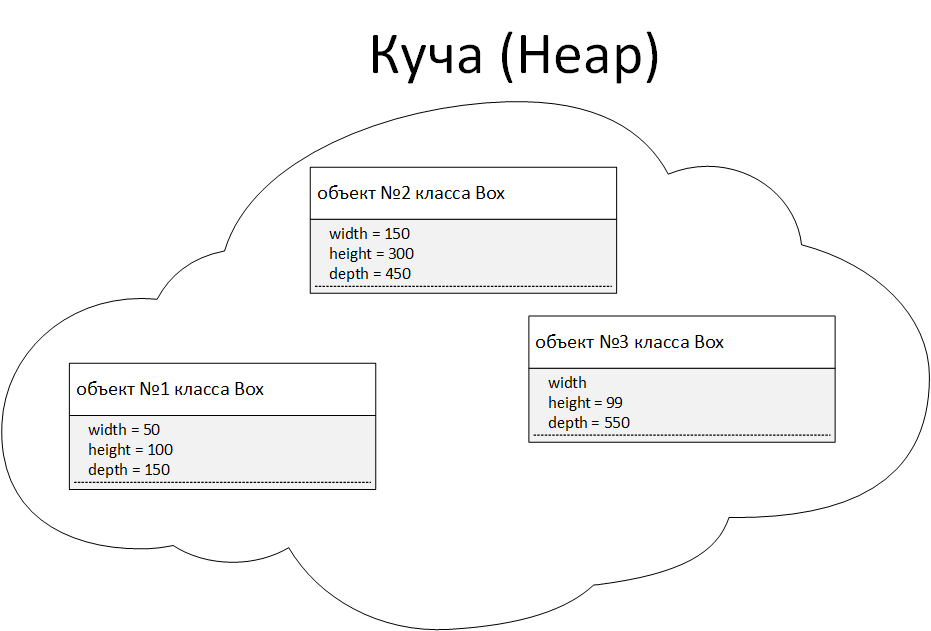
Как мы уже говорили, класс определяет тип данных. В данном случае новый тип данных называется Box. Это имя будет использовано для объявления типа Box. Не будем забывать, что объявление class создает только шаблон, но не конкретный объект. Таким образом, пока у нас есть только шаблон, и нет объектов типа Box.

Чтобы создать объект класса Box, необходимо воспользоваться оператором new. Оператор new создает экземпляр указанного класса и возвращает ссылку на созданный объект.



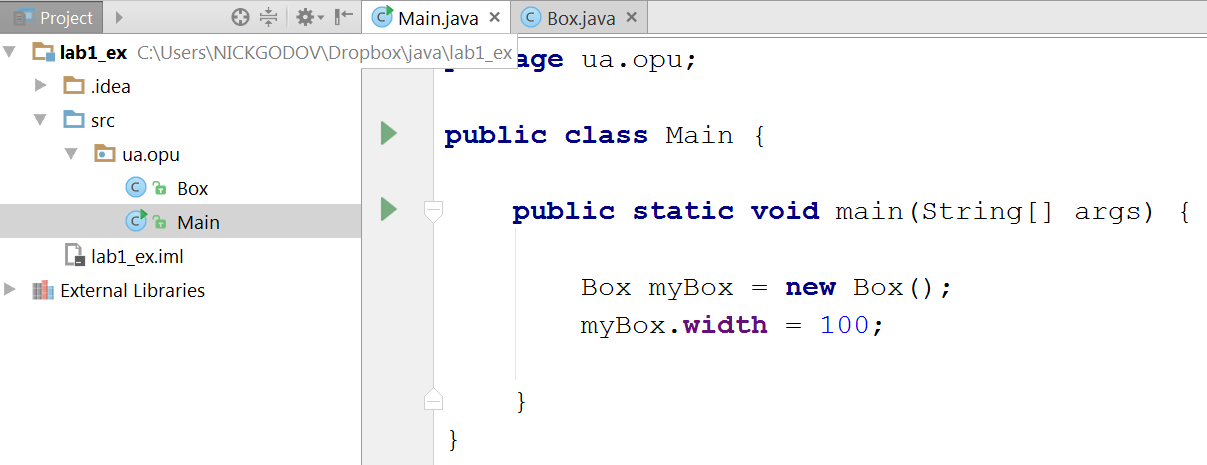
После выполнения этого оператора, объект ***myBox*** станет экземпляром класса ***Box***.

Напомним, что каждый объект содержит собственную копию переменной экземпляра, которая определена в классе. Таким образом, каждый объект класса Box будет содержать собственные копии переменных экземпляра ***width***, ***height***, ***depth***.



Изменения в переменных экземпляра одного объекта не влияют на переменные экземпляра другого объекта.

Для доступа к этим переменным служит операция-точка (**.**). Например, чтобы присвоить переменной ***width*** объекта ***myBox*** значение 100, нужно выполнить следующий код:



Этот оператор предписывает компилятору, что копии переменной ***width***, которая хранится в объекте ***myBox***, требуется присвоить значение 100. С помощью операции-точки можно получить доступ к переменным экземпляра и к методам в пределах объекта.

**ПРИМЕР НАПИСАНИЯ МЕТОДОВ КЛАССА**

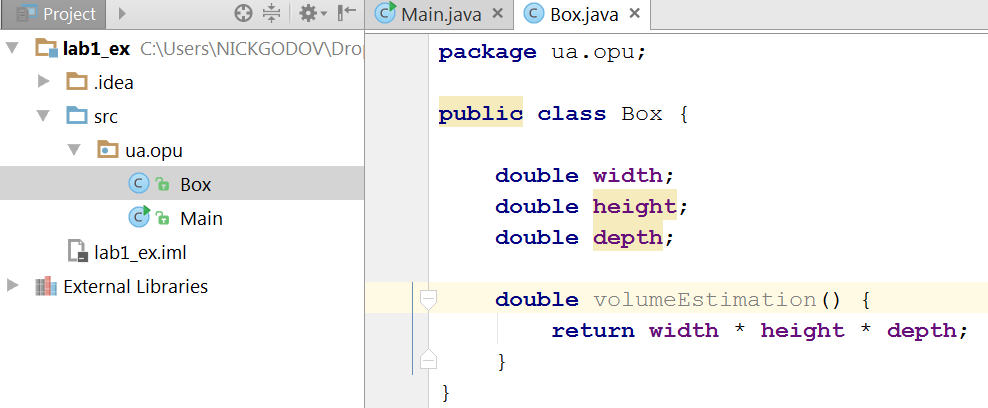
Общая форма объявления метода выглядит следующим образом:

|  |
| --- |
| тип имя\_метода(список параметров) {  //тело\_метода  } |

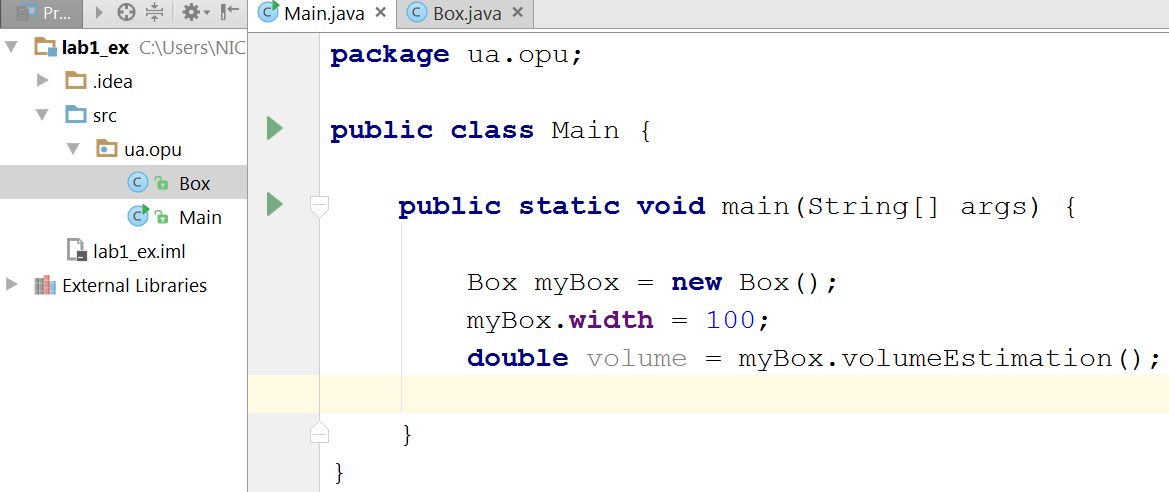
где тип обозначает конкретный тип данных, возвращаемый методом. Он может быть любым типом данных, в том числе и типом созданного класса.

Методы класса определяют интерфейс классов. Это позволяет тому, кто реализует класс, скрывать внутреннюю структуру класса и предоставлять вовне некоторый набор методов.

Вернемся к классу Box и добавим метод для вычисления объема коробки:



Вызовем этот метод у объекта ***myBox***, который мы создали ранее.



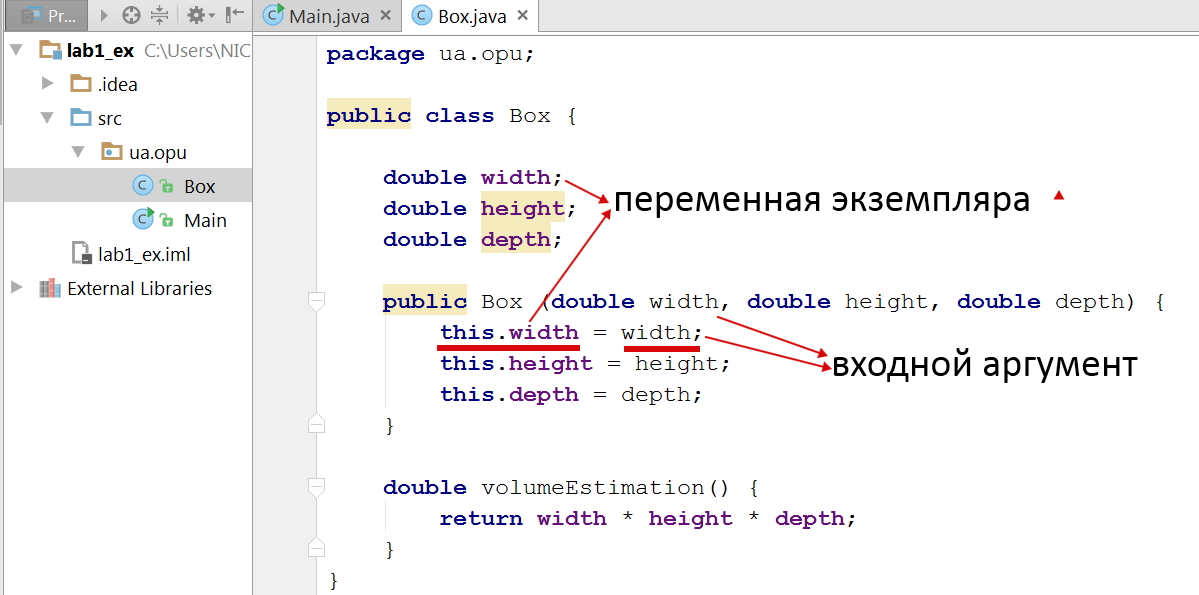
Обратите внимание, что метод ***volumeEstimation()*** был вызван по отношению к объекту ***myBox***. Это значит, что этот метод берет переменные ***width***, ***height*** и ***depth*** того объекта, у которого этот метод был вызван.

Также, следует обратить внимание, что когда мы описываем метод ***volumeEstimation()***, ссылка на переменные экземпляра ***width***, ***height*** и ***depth*** делается непосредственно без указания перед ними имени объекта или операции-точки. Когда в методе используется переменная экземпляра, определенная в его же классе, это делается непосредственно, без указания явной ссылки на объект и применения операции-точки. Это означает, что переменные экземпляра ***width***, ***height*** и ***depth*** неявно ссылаются на копии этих переменных, хранящиеся в объекте, который вызывает метод ***volumeEstimation()***.

**КОНСТРУКТОР**

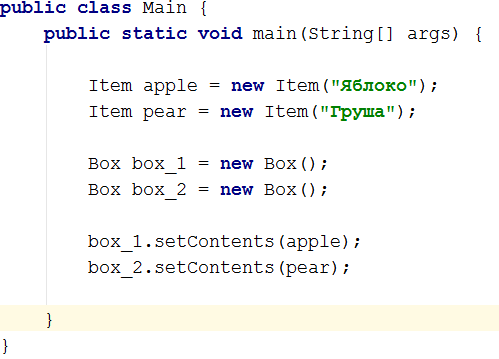
Инициализация всех переменных класса при каждом создании объекта – занятие довольно утомительное. В связи с этим, в Java разрешается выполнять собственную инициализацию при создании объектов. Такая инициализация осуществляется с помощью конструктора.

**Конструктор** – это специальный метод, который вызывается при создании нового объекта. Синтаксис конструктора отличается от синтаксиса обычного метода. Его имя совпадает с именем класса, в котором он находится, и он не имеет возвращаемого типа.



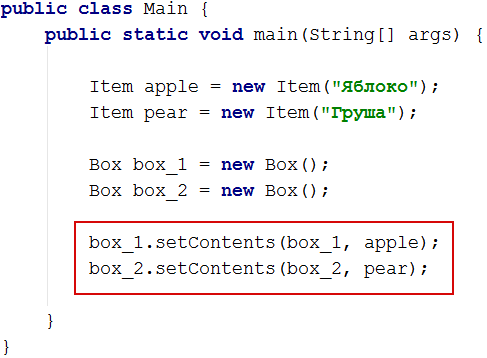
**КЛЮЧЕВОЕ СЛОВО THIS**

Представим, что у нас есть два объекта одного класса и для этих двух объектов вызывается один и тот же метод:

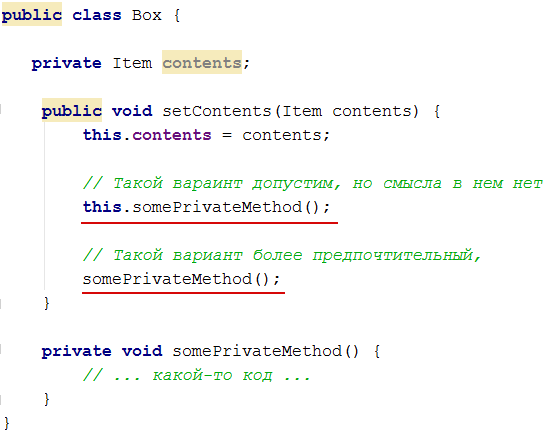


Если существует один метод ***setContents()***, как метод узнает, для какого объекта он вызывается – для ***box\_1*** или для ***box\_2***?

Оказывается, при вызове метода ***setContents()*** (как и при вызове любого другого метода) передается скрытый первый аргумент – ссылка на используемый объект. Таким образом, вызовы методов на самом деле выглядят так:

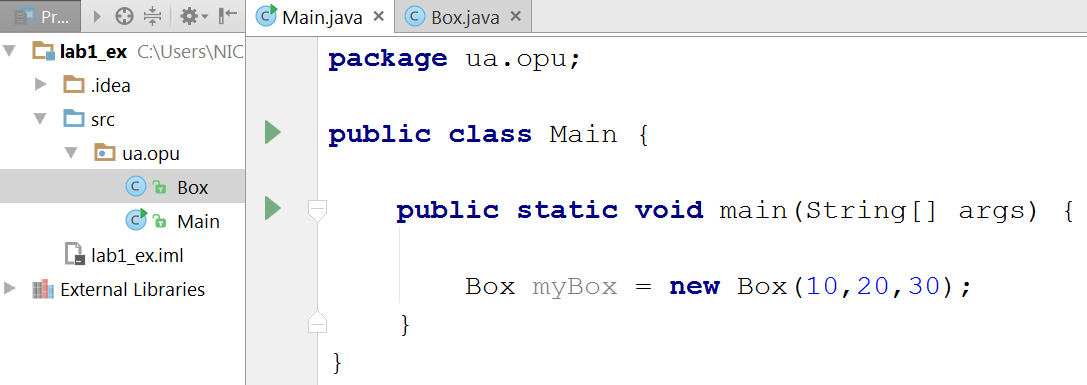


Иногда, во время выполнения метода необходимо получить ссылку на текущий объект, для которого был вызван метод. Так как ссылка на него передается скрытно, идентификатора для нее нет. Но для этого существует специальное ключевое слово – this. Ключевое слово ***this*** предоставляет ссылку на объект, для которого был вызван метод. Обращаться с ней можно как и с любой другой ссылкой на объект. Для вызова метода класса из другого метода этого же класса, использовать ключевое слово ***this*** не нужно.



В нашем случае, имена аргументов конструктора совпадают с именами переменных экземпляра, и, чтобы обратиться именно к экземплярам, мы используем ключевое слово **this** и с помощью оператора-точки обращаемся к переменной экземпляра и присваиваем ей новое значение.

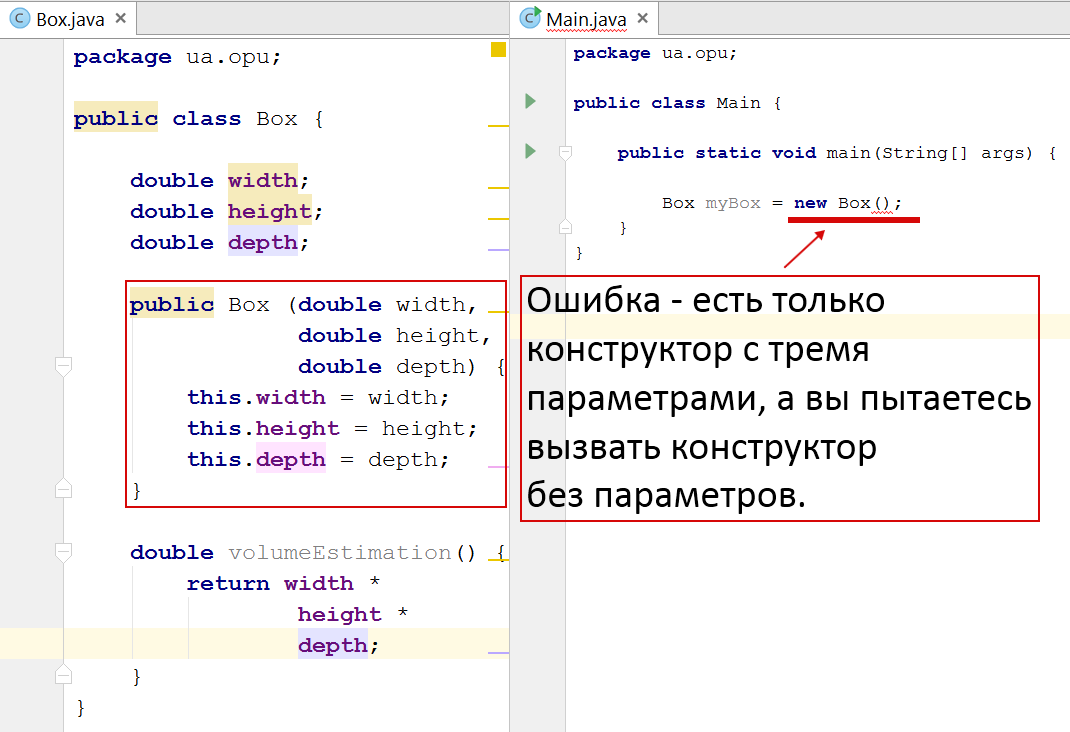
Теперь воспользуемся конструктором при создании объекта



Теперь нам должно быть понятно, почему при создании нового объекта, после имени класса требуется указывать круглые скобки. В действительности оператор new вызывает конструктор класса. Таким образом, в строке кода



Оператор new вызывает конструктор Box(). Но мы ранее не создавали этот конструктор, почему компилятор не выдал ошибку, когда мы запускали приложение? Если вы в классе не определите конструктор, Java создает в классе конструктор по умолчанию. Конструктор по умолчанию инициализирует все переменные значениями по умолчанию (***null*** для переменных, ссылающихся на объекты, ***false*** для переменных типа ***boolean*** и ***0*** для всех остальных примитивных типов). Если же вы определите в классе хотя бы один конструктор, то конструктор по умолчанию создан не будет. Именно поэтому, следующий код выдаст ошибку.



**ИНКАПСУЛЯЦИЯ И ОГРАНИЧЕНИЕ ДОСТУПА**

**Инкапсуляция** – один из основополагающих принципов ООП. Инкапсуляция – это одна из причин, почему так широко используется ООП.

Инкапсуляцию можно считать защитной оболочкой, которая предохраняет код и данные от произвольного доступа со стороны другого кода, находящегося снаружи оболочки. Доступ к коду и данным, находящимся внутри оболочки, строго контролируется тщательно определенным интерфейсом (набором общедоступных, публичных методов).

В любом классе присутствуют две части: интерфейс и реализация.

Интерфейс отражает внешнее поведение объектов этого класса. Внутренняя реализация описывает представления и механизмы достижения желаемого поведения объекта.

В интерфейсе собрано все, что касается взаимодействия данного объекта с другими объектами, а реализация скрывает от других объектов все детали, не имеющие отношения к процессу взаимодействия объектов.

**Инкапсуляция** – это процесс отделения друг от друга элементов объекта, определяющих его устройство и поведения; инкапсуляция служит для того, чтобы изолировать контрактные обязательства от их реализации.

Инкапсуляция позволяет локализовать части реализации системы, которые могут подвергнуться изменениям. По мере развития программы, разработчики могут принять решение изменить внутреннее устройство тех или иных объектов с целью улучшения производительности или экономии памяти. Но интерфейс будет нетронутым и позволит другим объектам таким же способом взаимодействовать с этим объектом. (Пример автомобиля – педали, руль, приборная панель и внутренняя начинка).

Инкапсуляция в Java реализована с помощью использования модификаторов доступа.

Язык Java предоставляет несколько уровней защиты, которые позволяет настраивать область видимости данных и методов. В Java имеется четыре категории видимости элементов класса:

* ***private*** – члены класса доступны только членам данного класса;
* по умолчанию (package-private) – члены класса доступны классам, которые находятся в этом же пакете;
* ***protected*** – члены класса доступны классам, находящимся в том же пакете, и подклассам – в других пакетах;
* ***public*** – члены класса доступны для всех классов в этом и других пакетах.

Член класса (переменная, конструктор, методы), объявленный ***public***, доступен из любого метода вне класса.

Всё что объявлено ***private***, доступно только конструкторам и методам внутри класса и нигде больше. Они выполняют служебную или вспомогательную роль в пределах класса и их функциональность не предназначена для внешнего пользования. Закрытие (***private***) полей обеспечивает инкапсуляцию.

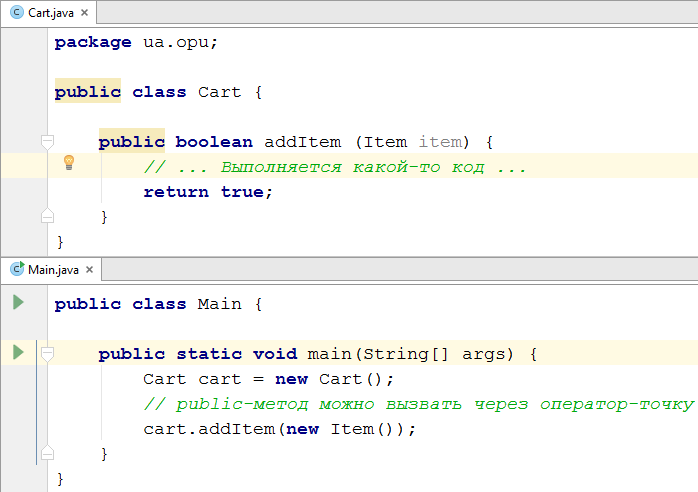
**ПРИМЕР ИСПОЛЬЗОВАНИЯ ИНКАПСУЛЯЦИИ**

Представим, что нам необходимо создать класс «Корзина» (Cart), который хранит в себе набор объектов класса «Товар» (Item).

Какие методы «Корзина» должна предоставлять для внешнего использования. Это могут быть, например, методы «Добавить товар», «Убрать последний добавленный товар», «Подсчет суммы цен товаров в корзине», «Повышение цен в корзине на N процентов» и «Снижение цен в корзине на N процентов».

|  |  |
| --- | --- |
| Название метода | Описание |
| public Cart(int capacity); | Конструктор с 1 параметром – максимальным количеством товаров в корзине |
| public boolean addItem(Item item); | Добавление товара в корзину. Возвращает успешность операции. |
| public Item deleteLastAddedItem(); | Удаление последнего добавленного товара в корзину. Возвращает удаленный товар. |
| public double calculateItemPrices(); | Подсчет суммы цен всех товаров в корзине. |
| public void raiseItemPrices(double percent); | Поднять цены товаров в корзине на определенный процент (значение процента передается как аргумент метода). |
| public void cutItemPrices(double percent); | Снизить цены товаров в корзине на определенный процент (значение процента передается как аргумент метода). |

Как вы можете заметить, это public-методы, а значит, их можно вызвать через оператор-точку имея ссылку ну объект.



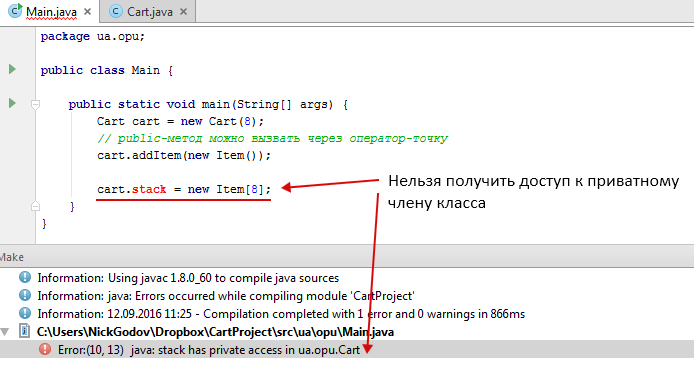
Перечень этих public-методов и составляет **интерфейс класса** – то есть, с помощью этих методов объект класса будет взаимодействовать с внешним миром. Эти методы имеют вполне четко определенные входные аргументы и могут возвращать значения четко определенных типов, и никак иначе. По аналогии с этим, поворот колес автомобиля осуществляется четко определенным образом – поворотом руля, и бензин надо заливать в четко определенное отверстие крышки бензобака, а не как-то еще.

То – как будет реализовано хранение товаров в корзине – это внутренняя логика класса и она не должна быть доступна внешнему миру, она должна быть скрыта от внешнего вмешательства. Другие классы, которые будут использовать объекты класса ***Cart*** не должны знать и не должны иметь доступ к тому – как там «внутри» реализовано хранение товаров, подсчет цен и изменение цены на определенный процент и т.д., они могут только лишь использовать предоставленные им ***public***-методы.

Давайте реализуем «Корзину» с помощью структуры «стек», которая, в свою очередь, реализована обычным массивом.

|  |
| --- |
| **public class** Cart {    **private** Item[] **stack**; *// массив для реализации стека*  **private int topIndex**; *// указатель на вершину стека*    *// При создании корзины мы должны*  *// указать максимальное количество элементов*  *// в корзине*  **public** Cart(**int** capacity) {  **stack** = **new** Item[capacity];  **topIndex** = -1;  }    *// Добавление нового товара в корзину*  **public boolean** addItem(Item item) {  **return** push(item);  }    *// Приватный метод, который реализует добавление в стек*  **private boolean** push (Item item) {  *// Добавляем товар в стек*  **return true**; *// или false если не стек переполнен*  }    *// Удаление последнего добавленного товара в корзину*  **public** Item deleteLastAddedItem() {  **return** pop();  }    *// Приватный метод, который реализует извлечение из стека*  **private** Item pop() {  **return new** Item(); *// Извлеченный из стека товар*  }  } |

Как мы видим, массив с товарами, указать на вершину стек объявлены как ***private*** члены класса. Это значит, что мы не можем получить к ним доступ извне – они доступны только внутри данного класса.



Программиста, который будет использовать класс ***Cart***, не должна волновать ситуация с переполнением стека, с попыткой извлечь элемент из пустого стека, он не должен следить за указателем на вершину стека, он даже не должен знать что это стек. Для него объект класса ***Cart*** это некоторый объект, который предоставляет «услугу» в виде корзины товаров и с этой корзиной можно работать с помощью определенных public-методов.

В дальнейшем мы можем переделать класс ***Cart*** и поменять внутреннюю реализацию. Мы можем использовать структуру очередь, мы можем использовать коллекции, мы можем иначе реализовать операции добавления и удаления элемента в стеке, но если мы сохраним интерфейс класса неизменным, то для внешнего мира эти изменения внутренней логики не будут важны и если мы поменяем внутреннюю логику одного небольшого участка программы, то вся остальная программа будет работать так же.

# ПРАКТИЧЕСКАЯ ЧАСТЬ

**ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ:**

1. Создайте класс «Товар» (Item).

А) Класс «Товар» должен содержать следующие поля: наименование (строка), цена (float).

Б) Класс «Товар» должен содержать конструктор, который принимает два параметра: наименование и начальную цену товара.

В) Класс «Товар» должен иметь следующие публичные методы: поднятие цены на определенный процент (значение процента типа float передается как аргумент метода); снижение цены на определенный процент (значение процента типа float передается как аргумент метода).

Г) В классе должна быть реализована проверка цены на отрицательное значение. Если в конструкторе передается отрицательное значение цены, либо в результате снижения цены на >100% цена становится отрицательной, она должна быть принудительно установлена в 0.

2. Создайте класс «Корзина» (Cart).

А) **«Корзина» должна реализовывать структуру данных стек**, в котором содержатся объекты класса «товар». Будем считать, что мы добавляем всегда по 1 единице товара. Стек «внутри» реализуется обычным массивом объектов Item. Класс должен содержать проверки, связанные с работой стека – переполнение стека, попытка извлечь элемент из пустого стека и т.д.

Б) «Корзина» должна содержать конструктор с 1 параметром – максимальным количеством элементов в стеке. В этом конструкторе происходила инициализация массива, который реализует стек.

В) «Корзина» должна содержать следующие публичные методы: добавление товара, удаление товара, подсчет суммы цен товаров в корзине (пройтись по элементам массива и сложить все значения цен), повышение и понижение цен всех товаров в стеке (два отдельных метода, значение цены передается как параметр метода. Необходимо пройтись по всем элементам массива и передать соответствующее сообщение объектам.

3. В методе Main необходимо создать объект класса «Корзина» с некоторым максимальным количеством элементов в стеке.

А) Заполнить корзину объектами класса Item (5-6 объектов будет достаточно);

Б) Вывести сумму цен товаров внутри корзины;

В) Поднять цены в корзине на 15%, вывести измененную сумму цен в консоль.

Г) Снизить цены в корзине на 30%, вывести измененную сумму цен в консоль.

4[[1]](#footnote-1). Перепишите класс Cart, используя вместо стека очередь (или коллекцию, если вы знаете, что это такое) без изменения интерфейса класса. Проверьте работу кода.

Код программы: package lab.pkg2.stytskovskyi;

/\*\*

\*

\* @author Student

\*/

public class Lab2Stytskovskyi {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args)

{

Product product = new Product("Soap", 3);

Product product1 = new Product("Crackers", 6);

Product product2 = new Product("Wine", 15);

Product product3 = new Product("Cornflakes", 7);

Product product4 = new Product("Toothpaste", 8);

ShoppingCart cart = new ShoppingCart(15);

cart.addProduct(product);

cart.addProduct(product1);

cart.addProduct(product2);

cart.addProduct(product3);

cart.addProduct(product4);

System.out.println("Total = " + cart.getPriceSum());

System.out.println("..............................");

cart.raisePrices(15);

System.out.println("Total(+15%) = " + cart.getPriceSum());

System.out.println("..............................");

cart.lowerPrices(30);

System.out.println("Total(+15% , -30%) = " + cart.getPriceSum());

}

}

package lab.pkg2.stytskovskyi;

/\*\*

\*

\* @author Student

\*/

public class Product

{

private String name;

private double price;

public Product (String n, double p)

{

name = n;

price = p;

}

public String getName() { return name; }

public String setName(String n) { name = n; return name; }

public double getPrice() { return price; }

public double setPrice(double p)

{

if (p >= 0)

price = p;

else

price = 0;

return price;

}

public void raisePrice (double percent)

{

price += price \* (percent/100);

}

public void lowerPrice (double percent)

{

price -= price \* (percent/100);

if (price < 0) { price = 0; }

}

}

package lab.pkg2.stytskovskyi;

import java.util.ArrayList;

/\*\*

\*

\* @author Student

\*/

public class ShoppingCart

{

ArrayList<Product> produce = new ArrayList<> ();

private int maxSize;

public ShoppingCart(int s) { maxSize = s; }

public int getMaxSize() { return maxSize; }

public int setMaxSize(int s)

{

if (s < 0)

s = 0;

maxSize = s;

return maxSize;

}

public void addProduct (Product p)

{

if (produce.size() < maxSize)

produce.add(0,p);

}

public void removeProduct ()

{

if (produce.size() > 0)

produce.remove(produce.size());

}

public double getPriceSum ()

{

double s = 0;

for (Product item:produce)

{

s += item.getPrice();

}

return s;

}

public void raisePrices (double percents)

{

double tempPrice;

for (Product item:produce)

{

tempPrice = item.getPrice();

tempPrice += tempPrice \* (percents / 100);

item.setPrice(tempPrice);

}

}

public void lowerPrices (double percents)

{

double tempPrice;

for (Product item:produce)

{

tempPrice = item.getPrice();

tempPrice -= tempPrice \* (percents / 100);

item.setPrice(tempPrice);

}

}

}

Результат работы программы:

1. - задание на дополнительные баллы [↑](#footnote-ref-1)