# Лабораторная работа №1

# КЛАССЫ И ОБЪЕКТЫ В JAVA

Цель: разработка консольных приложений использующих классы и объекты этих классов, изучить принцип инкапсуляции.

## Понятие классов и объектов в объектно-ориентированном программировании (ООП)

Попытаемся сначала понять в чем польза от классов и объектов.

Есть, к примеру, в базе некоторое количество автомобилей, у которых много общих параметров. Пытаясь задействовать эти данные в программе, очень неудобно заводить на каждый автомобиль свою отдельную ветвь переменных(количество дверей, объём двигателя, марка и т.д.). Поэтому было решено сделать комбинированный тип данных: «структуры» или «записи».

В разных языках и пособиях можно встретить оба названия, но смысл один: собрать вместе данные, характеризующие что-то общее.

Тип данных: Колесо {

переменная резина: Резина;

переменная диск: Диск;

}

Тип данных: Автомобиль {

переменная число\_дверей: Целое\_число;

переменная объем\_двигателя: Вещественное\_число;

переменная марка: Строка;

переменная модель: Строка;

переменная колесо: Колесо1;

переменная колесо: Колесо2;

переменная колесо: Колесо3;

переменная колесо: Колесо4;

...

}

Автомобиль авто1;

Автомобиль авто2;

Автомобиль авто3;

Рисунок 1.1 - Пример записи структуры

Из псевдокода выше можно заметить, что структура может в себя включать как простые поля, так и другие, ранее определённые, структуры, что очень удобно. После описания структуры «Автомобиль», в данном примере объявлены три переменные типа «Автомобиль»: авто1, авто2,авто3. Авто1 будет соответствовать синяя Рено, которая стоит в правом дальнем углу парковки, авто2 – желтая Мини, а авто3 – красная Мазда. Таким образом, можно описать все автомобили из базы данных, и тогда каждой переменной «авто» типа «Автомобиль» будет соответствовать одна машина на парковке. Теперь не нужно описывать каждую авто по отдельности, т.е. каждая переменная (авто1, авто2, авто3) сразу будет содержать перечень параметров: число дверей, объем двигателя и т.д. При этом переменные будут отличаться значениями этих параметров.

Далее этого стало недостаточно и было решено добавить в структуры поведенческий аспект. То есть, создать такую концепцию, при которой структурный тип не только содержал бы в себе все свои данные, но и сам бы с ними работал: загружал, сохранял, изменял и т.д.

Класс: Автомобиль {

...

переменная марка: Строка;

переменная модель: Строка;

...

функция заголовокДляПрайсЛиста() {

вернуть марка + " " + модель;

}

}

Автомобиль авто1;

Автомобиль авто2;

Автомобиль авто3;

Рисунок 1.2 – Пример класса

И это уже можно назвать классом (рисунок 1.2), хотя это, по сути, новый тип данных. Получается, **класс** — это некий шаблон, описание не только самой структуры данных (что там должно быть), но и поведения, функциональных возможностей объектов этого класса. Применительно к программированию, **объект класса** — это переменная этого класса, которая содержит конкретные данные и работает в соответствии с описанной в классе функциональностью.

В **теле класса** (между фигурными скобками, открывающая из котрых стоит сразу после имени класса «Автомобиль») описываются свойства будущих объектов — **поля**, содержащие данные, и **методы** — функции для каких-либо взаимодействий (с этими данными, с другими объектами, с окружением, с операционной системой и т.д.).

В данном псевдокоде авто1, авто2, авто3 являются объектами и помимо содержащих в себе данных, могут также возвращать строку, содержащую марку и модель, которую удобно использовать для прайс-листа.

И так, **класс = шаблон**. В реальной жизни можно взять в качестве примера чертёж какой-нибудь важной детали. Пусть в жизни и не так всё просто, и вместо одного чертежа, проект детали обычно является ворохом документации весом побольше, чем сама деталь из утяжелённого ураном чугуна, но мы всё же пока представим просто чертёж. Это и есть «класс» детали в программном смысле. Инженеру и рабочим на производстве достаточно иметь один чертёж, чтобы изготовить большое количество таких деталей — физических и конкретных воплощений класса (чертежа), объектов.

# Инкапсуляция.

Инкапсуляция (по-русски: «сокрытие») — это свойство объектов скрывать некоторые свои данные и способы их обработки (методы) от окружающей его цифровой среды и, в частности, от деятельности малоопытных программистов, оставляя «снаружи» только необходимые и/или требуемые свойства и функциональные возможности.

С точки зрения стороннего программиста объект выглядит этаким «чёрным ящиком», у которого есть входы и выходы. Подавая на входы данные и дёргая за рычажки (функциональные методы) можно получить какой-то результат на выходе.

Если вы мало что поняли из предыдущих двух абзацев, то вам надо сначала уяснить, что же это такое, **классы и объекты** из предыдущих пунктов. После чего можно продвигаться далее.

Это достигается следующим образом. Прописывается в классе объекта (класс — шаблон для объектов). На сегодняшний день выработано три основных уровня инкапсуляции, называемых модификаторами доступа: public, protected и private. Обычно эта «троица» имеется в каждом языке программирования, в котором реализована объектно-ориентированная концепция.

Существует понятие **«область видимости»** чего-либо в программе (функции, переменной, свойств и методов класса). Область видимости определяет, можете-ли вы обратиться с этого места программы к какой-либо конструкции, или нет. Данные модификаторы призваны регулировать видимость свойств и методов внутри и снаружи класса и объектов класса, т.е. каждому полю и каждому методу присваеватся какой-то из модификаторов доступа.

**Модификатор «private»**

Зачастую необходимо закрыть какой-то отлаженный работоспособный механизм от чужой деятельности. Незнакомый с вашим классом разработчик может случайно напортачить, вызвать не тот метод не в то время и получить с этого множество логических ошибок в программе. Виноватого, поверьте, он долго искать не будет — это вы. Поэтому, есть смысл «попрятать» критичные свойства (поля) и методы (функции класса) под модификатором «private». К полю или методу с модификатором «private» можно обратится только из тела класса.

**Модификатор «protected»**

Но «приватить» все подряд тоже нехорошо. Это резко ограничит возможные пути по модификации и расширению ваших классов при наследовании и, соответственно, более широкого их использования.

Поэтому и существует компромиссный вариант: protected — защищенный (метод или свойство). Как и приватные, защищенные конструкции не видны «снаружи» объекта. К полю или методу с модификатором «protected» можно обратится из тела класса и из тела дочерних классов. (подробнее в лабораторной работе №4)

**Модификатор «public»**

«Public» означает «публичный», «общедоступный». Свойства и методы объекта, объявленные в классе публичными, доступны «снаружи» объекта и являются как раз теми самыми входами, выходами и управляющими рычажками, с помощью которых разработчик данного класса позволил другим программистам управлять поведением объектов этого класса. К полю или методу с модификатором «public» можно обратится из любого места, в том числе из любого другого класса, где был объявлен объект.

При освоении нового для вас языка программирования, обязательно следует прочитать про особенности реализации ООП в целом и как сделано управление видимостью private-protected-public. Везде свои особенности. Например, в Java сами классы могут объединяться в пакеты классов и быть приватными или защищенными внутри пакетов. В итоге получаем дополнительный уровень доступа, дающий, с одной стороны, более широкие возможности по структуризации программы, а с другой стороны — дополнительные оттенки проектирования, которые надо учитывать.

**Модификатор доступа «friendly»**

В Java есть еще модификатор доступа **friendly.** Этот модификатор не пишется явно, но подразумевается, если не указан никакой другой модификатор доступа. При использовании friendly к данным может обратиться любой класс и метод в том же самом пакете.

Чтобы защитить поля от заведомо неправильных значений, их делают закрытыми, а доступ к ним осуществляется через общедоступные методы. При таком подходе в методе можно сделать проверку на то, соответствует ли полученное значение логическим требованиям (например, не пытается ли пользователь установить в поле число\_дверей отрицательное число) и при необходимости подкорректировать значение или запросить у пользователя другое, такой метод называется сеттером. **Сеттер** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) setter) - специальный метод, используемый в объектно-ориентированном программировании для того, чтобы присвоить какое-либо значение инкапсулированному (скрытому) полю.

Чтобы получить значение из инкапсулированного поля, нужен другой метод называемый геттером. **Геттер** (англ. getter — получатель) — (в программировании) специальный метод, позволяющий получить данные, доступ к которым напрямую ограничен. Это один из методов объектно-ориентированного программирования, может помочь реализовать гибкий механизм инкапсуляции.

Рассмотрим **пример создания простейшего класса**. Давайте с его помощью смоделируем окружности на координатной плоскости.

Каждая такая окружность, как известно, будет определяться своим центром (т.е. точкой с двумя числовыми координатами) и радиусом (т.е. его длиной, представляемой в виде числа). Таким образом, окружность на координатной плоскости характеризуют 3 вещественных числа. Значит в нашем классе должно быть три соответствующих свойства.

Наделим класс следующими возможностями: созданную на основе класса окружность должно быть возможно выводить на экран (в виде описания её характеристик), перемещать (т.е. совершать преобразование движения, меняя координаты её центра) и масштабировать (т.е. совершать преобразование подобия, меняя радиус окружности).

Пример 1.1 Описание отдельного нового класса

// описываем отдельный новый класс  
class Circle {  
    // свойства класса  
    private double x; // абсцисса центра  
    private double y; // ордината центра  
    private double r; // радиус  
    // методы класса  
    // выводит на экран параметры окружности  
    public void printCircle() {  
        System.out.println("Окружность с центром ("+x+";"+y+") и радиусом "+r);  
    }      
   // перемещает центр, движение окружности  
    public void moveCircle(double a, double b) {  
        x = x + a;  
        y = y + b;  
    }  
    // масштабирует, выполняет преобразование подобия с коэффициентом k  
    public void zoomCircle(double k) {  
        r = r \* k;  
    }

// устанавливает значения координат центра окружности (сеттер)  
    public void SetCenter(double a, double b) {  
        x = a;  
        y = b;  
    }

// устанавливает значения радиуса окружности (сеттер)  
    public void SetR(double a) {  
        r = a;    }  
    // возвращает значение координаты x центра окружности (геттер)  
    public double GetХ () {  
        return x;  
    }      
    // возвращает значение координаты y центра окружности (геттер)  
    public double GetY () {  
        return y;  
    }     
    // возвращает значение радиуса окружности (геттер)  
    public double GetR () {  
        return r;  
    }       
}

# Конструкторы

*Конструктор* – это метод класса, вызываемый при создании объекта. Он обладает некоторыми особенностями:

* имеет такое же название, как и класс;
* не возвращает никакого значения.

Когда мы создаём объект командой Circle o1 = new Circle(); используется так называемый конструктор по умолчанию (или конструктор без параметров) — это специальный метод класса, мы его не определяли явно, но даже если его не определить он создаётся автоматически, выполняется при создании каждого нового объекта и присваивает первоначальные значения его свойствам (инициализирует их). Значения по умолчанию для свойств зависят от их типа (0 или 0.0 для числовых типов, false для логического типа и т.д.).

От остальных методов конструктор отличается тем, что имеет то же самое имя, что и весь класс, а также не имеет типа возвращаемого значения (по сути, в результате своей работы конструктор возвращает новый, созданный им объект этого класса).

Пример 1.2 Конструктор

class Circle {  
    private double x; // абсцисса центра  
    private double y; // ордината центра  
    private double r; // радиус  
  
. . .

    // конструктор по умолчанию, теперь сразу после создания объекта будем  
    // получать окружность единичного радиуса с центром в начале координат  
    public Circle() {  
        x = 0.0;  
        y = 0.0;  
        r = 1.0;  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
      Circle o1 = new Circle();  
      o1.printCircle(); // Окружность с центром (0.0;0.0) и радиусом 1.0

o1.SetCenter(2.5, 3.0); // устанавливаем значения координат центра окружности х=2.5 и у=3.0

o1.SetR(3.5); // устанавливаем значения радиуса окружности равное 3.5     
      o1.printCircle(); Окружность с центром (2.5;3.0) и радиусом 3.5  
    }  
}

Нужно учитывать следующий факт:

# Объявление объектов и операция new

Получение объектов класса **−** это двухшаговый процесс. Во-первых, нужно объявить переменную типа «класс». Она не определяет объект. Это просто переменная, которая может ссылаться на объект. Во-вторых, нужно получить актуальную, физическую копию объекта и назначать ее этой переменной. Это можно сделать с помощью операции *new*. Операция *new* распределяет динамически (т.е. во время выполнения) память для объекта и возвращает ссылку на нее. Данная ссылка является адресом ячейки памяти, выделенной объекту вышеуказанной операцией. Затем эта ссылка сохраняется в переменной.

Ниже приведен пример, в котором создается объект типа *Box* двумя способами.

// первый способ

Box mybox = new Box();

// второй способ

Box mybox; // объявить ссылку на объект

mybox = new Box(); // распределить память для Вох-объекта

Первый вариант комбинирует два шага, как это было вначале описано, второй пошагово создает объект типа Box.

**Ключевое слово «this»**

Иногда будет требоваться, чтобы метод ссылался на вызвавший его объект. Чтобы это было возможно, в Java определено ключевое слово this. Оно может использоваться внутри любого метода для ссылки на текущий объект. То есть this всегда служит ссылкой на объект, для которого был вызван метод. Ключевое слово this можно использовать везде, где допускается ссылка на объект типа текущего класса.

Можно обойтись без this как мы это сделали в классе Circle, но если бы мы объявили локальную переменную в методе этого класса Circle с именем «r» , то обратится к полю радиуса из этого метода мы бы смогли только через ключевое слово this.

Пример 1.3 Использование ключевого слова this в сеттере

// описываем отдельный новый класс  
class Circle {  
    // свойства класса  
    . . .

    private double r; // радиус  
    // методы класса  
    . . .

// устанавливает значения радиуса окружности (сеттер)  
    public void SetR(double r) {  
       this.r = r;

}

. . .  
}

**Пример решения задачи.**

Создать несколько объектов класса согласно заданию. Значение полей задать с помощью сеттеров. В классе предусмотреть геттеры и метод, осуществляющий вывод на экран монитора соответствующую информацию об объекте.

Состав класса Drugstore (аптека): фамилия вла­дельца, наличие лицензии, месячная прибыль.

Пример 1.4 Пример решения задачи

package lab1;

public class Lab1 {

public static void main(String[] args) {

Drugstore d1 = new Drugstore();//создание объекта d1

Drugstore d2 = new Drugstore();//создание объекта d2

d1.setName("Nordin");

d1.setLicense(true);

d1.setProfit(1000);//заполнение полей объекта d1 с помощью сеттеров

String s = d1.getName();//получение значения поля name с помощью геттера

System.out.println("Информация об аптеке " + s);

d1.Print();//вывод информации об объекте d1

//с объектом d2 выполняем аналогичные действия

d2.setName("Kravira");

d2.setLicense(false);

d2.setProfit(2000);

s = d2.getName();

System.out.println("Информация об аптеке " + s);

d2.Print();

}

}

package lab1;

public class Drugstore {

private String name;

private boolean license;

private int profit;

public void setName(String name)// метод присваивает подпроцессу указанное в параметре имя{

this.name = name;

}

public void setLicense(boolean license)//аналогично с наличием лицензии {

this.license = license;

}

public void setProfit(int profit)//аналогично с суммой дохода {

this.profit = profit;

}

public Drugstore() {

}

public String getName()//метод возвращает строку с именем подпроцесса, установленным с помощью вызова setName {

return name;

}

public boolean isLicense() {

return license;

}

public int getProfit() {

return profit;

}

public void Print() {

System.out.println("Название: " + name);

if(license){

System.out.println("Есть лицензия.");

}else{

System.out.println("Нет лецензии.");

}

System.out.println("Доход: " + profit);

}

}

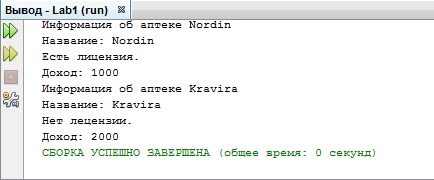


Рисунок 1.3 – Результат работы программы.

**Задания для самостоятельного выполнения**

Создать несколько объектов класса согласно заданию. Значение полей задать с помощью сеттеров. В классе предусмотреть геттеры и метод, осуществляющий вывод на экран монитора соответствующую информацию об объекте.

1. Состав класса Stock (склад): наименование това­ра, его стоимость, наличие товара на складе.

2. Состав класса Student (студент): фамилия, воз­раст, наличие задолженностей.

3. Состав класса Music (музыкальное произведение): название произведения, количество проданных копий, возможность скачать mp3 в интернете.

4. Состав класса Product (продукт): название про­дукта, наличие в магазине, цена.

5. Состав класса Book (книга): название, цена, нали­чие иллюстраций.

6. Состав класса Race (гонки): дата соревнования в виде символьной строки (год, месяц и день), количество участни­ков, наличие приза.

7. Состав класса Ward (больничная палата): количество мест, фамилия врача, наличие свободных коек.

8. Состав класса Tribe (племя): название, числен­ность. знакомство с огнем.

9. Состав класса Patient (пациент): фамилия, пол, возраст.

10. Состав класса Employee (служащий): фамилия, зарплата, наличие детей.

11. Состав класса Manager (управляющий): фами­лия, возраст, справляется ли с обязанностями.

12. Состав класса Icecream (мороженое): название, наличие шоколада, процент жирности.

13. Состав класса Person (индивидуум): фамилия, возраст, пол. Подсчитать средний воз­раст и количество мужчин.

14. Состав класса Butter (масло): название, наличие растительных добавок, цена.

15. Состав класса City (город): название, население, площадь.

16. Состав класса Milk (молоко): название, жирность в процентах, наличие на складе.

17. Состав класса Comicstore (магазин комиксов): фамилия вла­дельца, наличие лицензии, количество наименований продукции.

# Контрольные вопросы:

1. Что называется классом?
2. Перечислите члены класс.
3. Что нужно сделать для получения объектов класса?
4. Что произойдет, если в классе описан хотя бы один конструктор с параметрами?