**Лабораторная работа №2**

# МАССИВЫ. КОНСТРУКТОРЫ

Цель: Изучить механизм работы с конструкторами и освоить их перегрузку, научиться создавать массивы объектов класса.

**Понятие конструктора**

Конструктор — это метод класса, который вызывается в момент создания нового объекта, говорят инициализурует объект (**инициализация** – процесс присваивания значения полю в первый раз, в таком случае принято говорить, что поле инициализировано). Имя конструктора всегда**совпадает** с именем класса, в котором он расположен. У конструкторов не указывается тип возвращаемого результата (по сути, в результате своей работы конструктор возвращает новый, созданный им объект нужного класса). Часто в литературе это фраза упрощается и сводится к «у конструкторов нет типа возвращаемого результата (значения)».

Пример простого конструктора:

Пример 2.1 Пример простого конструктора

public class Film {

protected String title;

protected int views;

public Film(String t, int v){

this.title=t;

this.views=v;

}

}

Как видите, имя конструктора совпадает с именем класса. Этот конструктор выполняется при создании объекта Film, заполняя поля объекта заданными значениями.

Например, при создании объекта класса Film с помощью оператора:

new Film (“Beware of the car”,363);

поля экземпляра будут заполнены такими значениями:

title=” Beware of the car”;

views=363;

Между конструкторами и другими методами есть существенная разница: конструктор можно вызывать только в сочетании с операцией new.

Конструктор нельзя применить к существующему объекту, чтобы изменить информацию в его полях. Например, следующий вызов будет неправильным.

f1.Film(“Escape”,363); // ошибка

**Перегрузка конструкторов**

Мы рассмотрели, как можно писать конструктор, определяющий начальное состояние объектов. Однако создание объектов - очень важная задача, поэтому в языке Java предусмотрено много разнообразных механизмов написания конструкторов, с которыми мы познакомимся дальше.

Например, создать объект класса Film можно, как:

Film f1=new Film(“Beware of the car”,363);

То же можно сделать и следующим образом:

Film f1=new Film ();

Такая возможность называется перегрузкой (overloading).

Так как конструктор является методом, имеющим свои особенности, а методы можно перегружать, то для конструктора также можно использовать перегрузку.

О перегрузке говорят, если несколько конструкторов имеют одинаковое имя, но разные параметры.

Перегрузка является частным методом полиморфизма

**Полиморфизм**

Полиморфизм – возможность объектов с одинаковой спецификацией иметь различную реализацию. Кратко смысл полиморфизма можно выразить фразой: «Один интерфейс, множество реализаций».

Если перегруженные методы с одинаковыми именами находятся в одном классе, списки параметров должны отличаться. Но если метод подкласса совпадает с методом суперкласса (порождающего класса), то метод подкласса переопределяет метод суперкласса. Совпадать при этом должны и имена методов и типы входных и выходных параметров. В данном случае переопределение методов является основой концепции динамического связывания.

**Конструктор с параметрами**

В примере 2.1 мы рассмотрели конструктор, с полным набором параметром, т.е в классе Film нам нужно было задавать и название фильма, и количество зрителей, однако нам не всегда нужно знать все данные.

Рассмотрим такой пример: некоторый банк ведёт данные о своих клиентах. Значит нужен класс-клиент, и каждый конкретный клиент является объектом этого класса.

Таблица 2.1 Данные о клиентах некоторого банка

|  |  |  |  |
| --- | --- | --- | --- |
|  | клиент 1 | клиент 2 | клиент 3 |
| Фио клиента | Семёнов Аркадий Павлович | Щербаков Никита Геннадьевич | Жимайлов Иван Михайлович |
| Паспортные данные | BM2304117 | MA1601223 | BM0101123 |
| Заработок | 1000$ | - | 2000$ |
| кол-во поручителей | - | 2 | 3 |

Конструктор используется в момент оформлления кредита.Для оформления кредита на большую сумму (например на квартиру) банку нужно знать все данные (ФИО, паспортные данные, заработок, количество поручителей), однако для того же кредита, но на значительно меньшую сумму, банку уже не надо, чтобы за вас кто-то поручался, т.е в этом случае надо два конструктора: с четырьмя и с тремя параметрами. Тогда говорят, что конструктор с четырьмя параметрами (с полным набором параметров, т.е заполняем весь объект значениями) будет перегружаться конструктором с тремя параметрами (с неполным набором параметров, т.е заполняем не весь объект значениями).

Пример 2.3 Перегрузка конструктора с параметрами для класса Bank

public class Bank{

protected String fio

protected int income;

protected String passport\_details;

protected int number\_of\_guarantors;

public Bank(String f, int I, String p, int n){ // конструктор с полным набором парметров

this.fio=f;

this.income =i;

this.passport\_details=p;

this.number\_of\_guarantors=n;

}

public Bank(String f, int I, String p){ // конструктор с неполным набором параметров (с заданными ФИО, доходом и паспортными данными)

this.fio=f;

this.income =i;

this.passport\_details=p;

}

**Конструктор по умолчанию**

Конструктор по умолчанию позволяет установить всем полям соответствующие стандартные значения, после размещения нового объекта в памяти. Он не имеет параметров. Для класса Film конструктор по умолчанию имеет вид:

Пример 2.2 Конструктор по умолчанию для класса Film

public Film {

title=” ”;

views=0;

}

Если в классе не определяются конструкторы, то создаётся конструктор по умолчанию. Этот конструктор присваивает все полям экземпляра их значения, предусмотренные по умолчанию. Но если в классе описан явно хотя бы один конструктор с параметрами, то конструктор по умолчанию (без параметров) создаваться автоматические уже не будет (его в такой ситуации надо описывать явно). Хотя, если вам требуется только конструктор с параметрами (как второй из нашего примера), то можно обойтись и совсем без конструктора по умолчанию (описать в классе только один конструктор с параметрами).

**Конструктор копирования**

Конструктор копирования применяется для копирования информации выбранного объекта, часто используется для того, чтобы ещё раз не вводить данные.

Пример 2.4 Конструктор копирования для класса Film

public Film(Film object){

this.title=object.title;

this.views=object.views;

}

Копирующий конструктор, как и следует из названия, создаёт один новый экземпляр, инициализируя его значениями из старого объекта.

Мы рассмотрели: какие бывают конструкторы, изучили принцип их работы, а теперь кратко осветим основные моменты:

* Имя конструктора совпадает с именем класса.
* Класс может иметь несколько конструкторов.
* Конструктор может иметь ноль, один или более параметров.
* В конструкторе не указывается возвращаемое значение.
* Конструктор всегда вызывается совместно с операцией new

**Статические методы и переменные**

Объявляются при помощи ключевого слова *static*.

Переменные и методы, объявленные как *static*, относятся ко всему классу в общем. То есть одна копия переменной разделяется всеми экземплярами объектов класса, поэтому к ним можно обращаться без ссылки на конкретный объект без создания каких-либо объектов.

Для вызова статистической переменной, вы должны указать имя класса, точку и имя переменной. Для вызова статического метода надо будет сделать нечто подобное, только дополнительно, если метод передаёт что-то, в скобках надо передать этот тип .Статические переменные объявляют, когда нужна общая переменная на все экземпляры класса. Это может быть, например, счетчик количества экземпляров класса.

Пример 2.4 Счётчик количества экземпляров

class someclass{

static public int number = 0; //в начале выполнения программы объектов нет, значит счетчик =0

public someclass(){ // конструктор, в котором увеличивается значение счётчика, при каждом вызове

number++;

}

public static void main(String args[]){

System.out.println("Количество экземпляров класса someclass" + someclass.number); // вывод количества экземпляров класса

someclass z1 = new someclass();// создаётся объект класса someclass

someclass z2 = new someclass();

System.out.println("Количество экземпляров класса someclass" + someclass.number); // вывод количества экземпляров класса

}

}

Результат выполнения программы:

Количество экземпляров класса someclass 0

Количество экземпляров класса someclass 2

Методы, объявленные как *static*, можно вызвать, если объект не создан. В статических методах нельзя ссылаться на *this* и на *super* (подробнее в лабораторной работе №4). Такие методы могут работать только с переменными и методами, объявленными как *static*.

Cтатические переменные создаются один раз, как глобальные, то есть нет дублирования, как при нестатических переменных. Статические переменные можно вызывать, не создав объекта класса.

Пример 2.5 Изменение состояния статической переменной

class Myclass

{

static boolean b = true;

static int a = 1;

public static void f1(int x) {

if(ab\_switch){

System.out.println("Значение b истинно");

}else{

System.out.println("Значение b ложно");

}

System.out.println("a = "+a);

a = x;

System.out.println("a = "+a);

b =! b;

}

}

class Main

{

public static void main(String args[]) {

Myclass.f1(25);

Myclass.f1(20);

}

}

В данном примере b – изменяемая переменная, и вызов метода *f1* из любого метода меняет состояние *b* для всех экземпляров объектов, в которых эта переменная определена.

Результат выполнения программы:

Значение b истинно

a = 1

a = 25

Значение b ложно

a = 25

a = x;

Метод *main()* объявляется как *static*, потому что должен вызваться прежде, чем будут созданы какие-либо объекты.

*main()* – главная функция, с её начинается выполнение любой программы. Она вызывается кодом, который находится вне программы (поэтому *main( )* объявляется как *public*).

**Пакеты**

Язык Java позволяет объединять классы в наборы, называемые пакетами (package). Пакеты облегчают организацию работы и позволяют отделить классы, созданные одним разработчиком, от классов, разработанных другими.

Каждый пакет, как и папка, имеет свое имя, притом в Java реально создаются папки под ваши пакеты с совпадающим именем.

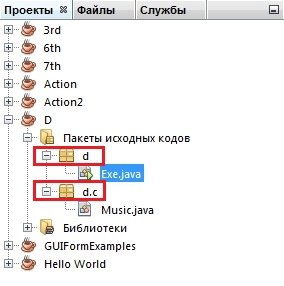


Рисунок 2.1 Представление пакетов в среде NetBeans

Как видим из рисунков, пакеты действительно схожи с папками. В данном случае представлена вложенность классов по пакетам.

Стандартная библиотека языка Java содержит большое количество пакетов: в качестве примеров можно привести java.lang, java.util, java.net и т.д. Стандартные пакеты языка Java представляют собой иерархические структуры. Подобно каталогам на диске, пакеты могут быть вложены один в другой. Все стандартные пакеты, принадлежат иерархиям java и javax.

**Импортирование классов**

Класс может использовать все классы из собственного пакета и все *общедоступные* (с модификатором доступа public) классы из других пакетов. Доступ к классам из других пакетов можно получить с помощью ключевого слова import. Например:

import ИмяПакета.ИмяКласса;

import ИмяПакета.\*;

В первом случае мы импортируем только один класс, а во втором — импортируем полный пакет.

Основным моментом также является то, что, если в двух пакетах содержатся классы с одинаковым именем и вы захотите импортировать их, например, в главном, то может возникнуть ошибка так как компилятор не сможет определить, какой класс именно нам нужен в данный момент.

**Модификаторы классов**

Так как модификатор относится к классу, то он прописывается непосредственно перед его объявлением:

Например, для класса Film это будет выглядеть следующим образом:

public class Film {

}

Как было сказано выше, для того чтобы вызвать класс из другого пакета нужно чтобы он был общедоступным (т.е public-доступным для использования вне зависимости от пакета или каталога) , однако этим модификаторы не заканчиваются, ниже представлен весь список, который используется в Java.

*friendly* – модификатор класса по умолчанию, если модификатор не определен явно для класса. Такой класс доступен только для объектов, находящихся в том же пакете. Вне пакета он выступает как *private*.

*final* – класс не может быть наследован, т.е. иметь подклассов.

*abstract* – класс, в котором объявлен хотя бы один абстрактный метод. Для таких классов нельзя создавать объекты. Такие классы используются для производных классов.

**Массивы**

**Одномерные массивы**

Как вы знаете из СИ, массив-это структура данных, в которой хранятся величины одинакового типа. Доступ к отдельному элементу массива осуществляется с помощью целочисленного индекса. Например, если a –массив целых чисел, то значение выражения a[i] равно i-му целому числу в массиве.

Для объявления массива вы можете пользоваться квадратными скобками, расположив их справа от имени массива или от типа объектов, из которых составлен массив, например:

int array[];

int[] array;

Допустимы оба варианта, поэтому вы можете выбрать тот, который вам больше нравится. Однако так мы только объявляем переменную a, не инициализируя её. Чтобы создать массив, нужно применить операцию new.

int []array =new int[100];

Приведенный оператор создаёт массив, состоящий из 100 целых чисел. При создании массива чисел все элементы его инициализируются нулями.

В языке Java также можно одновременно создать массив и инициализировать. Пример такой синтаксической конструкции:

int []array ={1,5,7};

Массивы объектов

Одномерный массив объектов — это массив ссылок на объекты. Соответственно, нужно создать как массив, так и сами объекты. Наиболее частая ошибка у начинающих при работе с массивами классов примерно следующая. Создается сам массив, например,

Film[] f1 = new Film[10];

а потом сразу идет попытка работы с элементами этого массива. Но здесь построен только массив ссылок, а сами объекты еще не созданы!

Пусть у нас есть некоторый класс Film и нужно построить массив из 4-x объектов этого класса.

Вариант 1. (явное создание)

Film f[] = new Film[4];

for (int i = 0; i < 4; j++ )

f[i] = new Film();

Вариант 2. (использование списка инициализации)

Film f [] = new Film[4] {

f1[0] =new Film(), // создание объекта с помощью конструктора по умолчанию

f[1]=new Film (“Avatar”,261), // создание объекта с помощью конструктора с полным набором параметров

f[2]=new Film (261), // создание объекта с помощью конструктора с неполным набором параметров

f[3]= new Film (f1[0]), // создание объекта с помощью конструктора копирования};

**Пример решения задачи.**

Создать массив объектов класса согласно заданию. Инициализацию значений полей осуществить с помощью различных конструкторов. В классе предусмотреть геттеры, метод, осуществляющий вывод на экран монитора соответствующую информацию об объекте и перегруженные конструкторы. Предусмотреть статические методы, подсчёта общей прибыли и количества аптек с лицензией. Главный класс с методом main должен находиться в другом пакете.

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

**package** lab;

**import** lab2.Drugstore;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Drugstore []d1 = **new** Drugstore[4]; //создаём массив из 10 объектов класса Drugstore

d1[0] = **new** Drugstore("Kolenko", **true**, 100); //создаём первый объект массива с помощью конструктора с полным набором параметров

d1[1]=**new** Drugstore("Petrov",**false**); // аналогично заполняем другой, но теперь пользуемся конструктором без одного параметра

d1[2]=**new** Drugstore(); // заполняем третий с помощью конструктора по умолчанию, все поля будут проинициализированы нулями

d1[3]=**new** Drugstore(d1[0]); // вызов конструктора копий, здесь мы копируем информацию, из первого объекта

**for**(**int** i=0;i<4;i++){

d1[i].print(); }// выводим полученную информацию.

System.***out***.println("sum of profits "+ Drugstore.*Profit*(d1)); //выводим суммы зарплаты через вызов статистического метода

System.***out***.println("owners with license "+ Drugstore.*hasLicense*(d1));

//выводим количество владельцев с лицензиями через вызов статистического метода

} }

**package** lab2;

**public** **class** Drugstore {

**private** String name; //фамилия

**private** **boolean** license; //лицензия. переменная типа булиан для того, что Вам нужно лишь знать: есть она или нет. То есть true или false. Булиан принимает только эти два значения тру или фолс.

**private** **int** profit; //обычное число: доход

**public** Drugstore() { //конструктор без параметров(по умолчанию). вызовется если будет создан объект таким образом :

} //Drugstore d = new Drugstore(); то есть не будут переданы параметры.

//если Вы для такого объекта попробуете распечатать информацию о нём - Вам выдаст значения по умолчанию, т.е все поля проинициализирует нулями, либо неизменяемыми значениями

//которые вы инициализировали в начале

**public** Drugstore(String name, **boolean** license, **int** profit) { //конструктор с полным набором параметров. То есть переданными значениями будет инициализирован объект

**this**.name = name;

**this**.license = license;

**this**.profit = profit;

}

**public** Drugstore( Drugstore o){ // создаине конструктора копирования

**this**.name=o.name;

**this**.license=o.license;

**this**.profit=o.profit;

}

**public** Drugstore(String Name, **boolean** License ){ // конструктор с неполным набором параметров, здесь инициализируется только имя и наличие лицензии

**this**.name=Name;

**this**.license=License;

}

**public** String getName() { //методы получения полей объекта.

**return** name;

}

**public** **boolean** isLicense() {

**return** license;

}

**public** **int** getProfit() {

**return** profit;

}

**public** **static** **int** Profit(Drugstore []drugstore){ //статический метод подсчёта прибыли, передаём массив Drugstore для подсчёта 4-х объектов

**int** p=0;

**for**(**int** i=0;i<4;i++){

p+=drugstore[i].profit; //складываем все зарплаты

**return** p;

}

**public** **static** **int** hasLicense(Drugstore []drugstore){

**int** lic=0;

**for**(**int** i=0;i<4;i++)

**if**(drugstore[i].license==**true**) //если владелец имеет лицензию, то увеличиваем счётчик

lic++;

**return** lic;

}

**public** **void** print() { //метод для вывода введенной информации

System.***out***.println(**this**.getName()+"\n"+

**this**.getProfit());

**if**(**this**.isLicense()){

System.***out***.println("Have license.");

}else{

System.***out***.println("Have not license.");

}

}

}

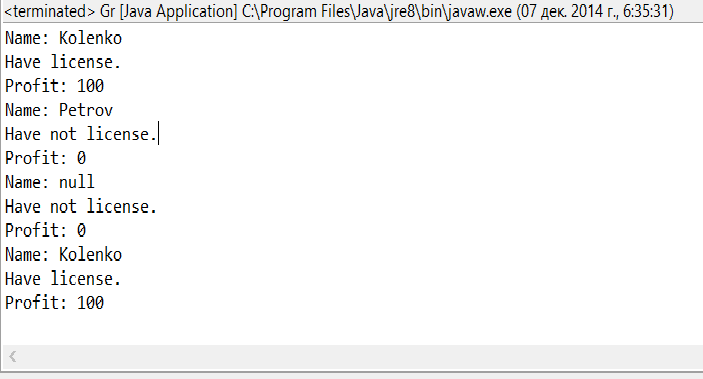


Рис 2.3 Результат работы программы

**Задания к лабораторной работе №3**

Создать массив объектов класса согласно заданию. Инициализацию значений полей осуществить с помощью различных конструкторов. В классе предусмотреть геттеры, метод, осуществляющий вывод на экран монитора соответствующую информацию об объекте и перегруженные конструкторы.

Предусмотреть статические методы подсчёта данных(на основе индивидуального задания). Главный класс с методом main должен находиться в другом пакете.

1. Состав класса Stock (склад): наименование това­ра, его стоимость, наличие товара на складе. Подсчитать общую стоимость товаров и количество товаров в наличии.

2. Состав класса Student (студент): фамилия, воз­раст, наличие задолженностей. Подсчитать средний воз­раст студентов и количество студентов с задолженностями.

3. Состав класса Music (музыкальное произведение): название произведения, количество проданных копий, возможность скачать mp3 в интернете. Подсчитать общую количество проданных копий и количество произведений, которые можно скачать mp3 в интернете.

4. Состав класса Product (продукт): название про­дукта, наличие в магазине, цена. Подсчитать общую стоимость продуктов и количество продуктов в наличии.

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

Подсчитать общую стоимость книг и количество книг с иллюстрациями.

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

7. Состав класса Ward (больничная палата): количество мест, фамилия врача, наличие свободных коек. Подсчитать общее количество мест и количество палат с свободными койками.

8. Состав класса Tribe (племя): название, числен­ность. знакомство с огнем. Подсчитать общую числен­ность и количество племен знакомых с огнем.

9. Состав класса Patient (пациент): фамилия, пол, возраст. Подсчитать средний воз­раст и количество женщин.

10. Состав класса Employee (служащий): фамилия, зарплата, наличие детей. Подсчитать среднюю зарплату и количество служащих с детьми.

11. Состав класса Manager (управляющий): фами­лия, возраст, справляется ли с обязанностями. Подсчитать средний воз­раст управляющих и количество управляющих, которые справляются с обязанностями.

12. Состав класса Icecream (мороженое): название, наличие шоколада, процент жирности. Подсчитать средний процент жирности и количество мороженых с шоколадом.

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

14. Состав класса Butter (масло): название, наличие растительных добавок, цена. Подсчитать общую стоимость масла и количество видов масла с растительными добавками.

15. Состав класса City (город): название, население, площадь. Подсчитать общую площадь и количество городов миллионеров.

16. Состав класса Milk (молоко): название, жирность в процентах, наличие на складе. Подсчитать средний процент жирности и количество видов молока на складе.

17. Состав класса Comicstore (магазин комиксов): фамилия владельца, наличие лицензии, количество наименований продукции. Подсчитать среднее количество наименований продукции по всем магазинам и количество магазинов с лицензией.

**Контрольные вопросы**

1. Что такое конструктор?
2. Чем конструктор отличается от метода?
3. Как создать массив объектов?
4. Для чего нужен оператор new?
5. Для чего используются пакеты?
6. Какие бывают конструкторы?
7. Как вызвать статистический метод?