МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Белорусский национальный технический университет

О. М. Болбот
В. В. Сидорик

КЛАССЫ В ЯЗЫКЕ ПРОГРАММИРОВАНИЯ JAVA

Учебно-методическое пособие
для студентов и слушателей системы повышения квалификации
и переподготовки

Под общей редакцией *В. В. Сидорика*

Рекомендовано учебно-методическим объединением
в сфере высшего образования Республики Беларусь

Минск
БНТУ
2020

УДК 004.483(075)
ББК 32.973.26-018.1я7
Б79

Р е ц е н з е н т ы:
доцент кафедры «Электронные вычислительные машины»
Белорусского государственного университета
информатики и радиоэлектроники, доцент
канд. физ.-мат. наук *И. И. Иванов*;
доцент кафедры «Физика и информатика»
Белорусского государственного педагогического
университета им. Максима Танка, доцент
канд. физ.-мат. наук *Г. А. Заборовский*

Болбот, О. М.

Б79 Классы в языке программирования Java : учебно-методическое пособие для студентов и слушателей системы повышения квалификации и переподготовки / О. М. Болбот, В. В. Сидорик ; под общ. ред. В. В. Сидорика. – Минск : БНТУ, 2020. – 76 с.

ISBN 978-985-550-895-4.

В учебно-методическом пособии описывается использование классов в популярном языке программирования Java в среде NetBeans 7.4. Рассматривается структура класса, создание экземпляров класса, принципы инкапсуляции, полиморфизма, наследования и композиции. Приводятся основные сведения о правилах именования в Java, способах комментирования программного кода. Затрагиваются вопросы построения UML-диаграмм классов.

Предназначено для студентов, слушателей системы повышения квалификации и переподготовки, преподавателей.

УДК 004.483(075)
ББК 32.973.26-018.1я7

ISBN 978-985-550-895-4

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В пособии описывается использование классов в популярном языке программирования Java: рассматривается структура класса, назначение членов класса, создание экземпляров класса, описываются модификаторы доступа для членов класса, изучаются принципы инкапсуляции, полиморфизма, наследования и композиции.

В пособии также приводятся основные сведения о правилах именования в Java, способах комментирования программного кода, затрагиваются вопросы построения UML-диаграмм классов.

Особенностью данного материала является поэтапное изучение основных принципов объектно-ориентированного программирования на примере одной сквозной задачи. Разработка классов и отладка программного кода производились в среде NetBeans 7.4, поэтому в приложении приводятся основные сведения и приемы работы, необходимые для выполнения практических упражнений.

Рассматриваемая задача затрагивает сферу издательства, выпускающего печатную продукцию (книги, учебники, журналы и т. п.). Работники издательства имеют дело с авторами материалов и ведут учет издаваемой продукции. В рамках тематики данного пособия поэтапно разрабатываются классы приложения, моделирующие такие объекты, как статья, книга, автор, журнал, а также организовывается взаимодействие между этими классами.

1. КЛАСС. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Определение класса. Структура класса

Язык программирования Java позволяет создавать классы, моделирующие объекты реального мира, которые могут быть как одушевленными, так и неодушевленными. Любой объект характеризуется определенными свойствами, такими как цвет, размер, масса, цена и т. п. Например, можно создать класс *Студент* или *Книга* и определить им некоторые свойства. Так как свойств у реального объекта может быть довольно много, то при описании классов используются те свойства, которые важны для решения данной конкретной задачи. Если необходимо создать приложение для автоматизации работы деканата, то для класса *Студент* логично выделить такие свойства, как фамилия, имя, отчество, курс, группа и т. д. А если нужно разработать приложение для автоматизации работы библиотеки, то для класса *Книга* будут важны такие свойства, как название книги, автор, издательство, год выпуска, количество страниц и т. п. Эти свойства называются полями класса (*data field*).

Объекты реального мира производят какие-либо действия (или над ними производят действия). Например, автомобиль движется, студент сдает сессию, компьютер производит вычисления. В языке Java для описания подобных действий имеются средства, которые называют методами (*methods*). Например, если машина движется с заданной скоростью, то можно вычислить путь, пройденный ею за определенное время; для студента можно вычислить средний балл, а для книги можно рассчитать количество авторских листов.

Класс описывает абстрактный объект реального мира, то есть любого студента или любую книгу. Для работы с конкретными объектами в языке программирования Java существуют средства, с помощью которых можно создать экземпляры класса, например, студента второго курса машиностроительного факультета БНТУ Петрова Владимира Михайловича; конкретные книги: например, книгу Монахова «Язык программирования Java» или книгу Эккеля «Философия Java». Все экземпляры одного класса имеют одинаковые наборы полей (свойств, характеристик), но значения этих полей для каждого экземпляра – свои. Значения полей определяют состояние экземпляра класса, а методы – его поведение.

Класс (*class*) – это шаблон для создания объектов с заданными свойствами и методами. Все экземпляры одного класса (объекты, порожденные от одного класса) имеют один и тот же набор свойств и общее поведение. Класс содержит описание полей (свойств), характеризующих объект, и методы, описывающие его поведение.

Поля и методы, описанные в классе, называют членами класса (*class members*).

Экземпляры класса являются «воплощением» в реальность того, что описано в классе.

Правила именования классов, полей, методов

Как правильно записывать имена полей, методов класса?

Любая последовательность букв, цифр и знаков подчеркивания, если она начинается с буквы, с точки зрения синтаксиса языка подходит на роль имени. Но для облегчения понимания программного кода желательно соблюдать соглашения кодирования Java и создавать составные осмысленные имена.

При выборе имени необходимо руководствоваться следующими правилами:

– для имени следует выбирать конкретные слова, которые характеризуют данный объект;

– все имена следует записывать по-английски: fileName, но не imyaFayla;

– имена полей и классов должны быть существительными, а имена методов – глаголами;

– имена классов должны начинаться с прописной буквы; если в его имени несколько слов, они записываются в смешанном регистре, начиная с верхнего: Line, MyInterface;

– имена полей класса должны начинаться со строчной буквы; если в его имени несколько слов, то они записываются в смешанном регистре, начиная с нижнего: line, borderColor;

– именованные константы должны быть записаны в верхнем регистре с нижним подчеркиванием в качестве разделителя: MAX_ITERATIONS, COLOR_RED, PI;

– имена методов должны начинаться со строчной буквы; если в имени метода несколько слов, то они записываются в смешанном регистре, начиная с нижнего: getName();

– можно использовать суффиксы и префиксы для добавления дополнительной информации к имени;

– префикс is следует использовать только для булевых (логических) полей и методов: isVisible, isFound, isOpen.

– имя объекта должно соответствовать его содержанию, например: MaxItem – максимальный элемент, NextItem – следующий элемент.

Исходя из вышеописанных правил, класс *Книга* должен называться Book, его поля: название книги – bookTitle, автор – author, издательство – publishingHouse, год выпуска – year, количество страниц – pageCount, метод для вычисления количества печатных листов – calculateCountPrintedSheets.

Типы полей

Значения полей класса *Книга* являются различными величинами: название, автор, издательство – это текстовые величины; год выпуска и количество страниц – целые числа; метод вычисления количества печатных листов получает в результате дробное число. В языке программирования Java поля класса реализуются с помощью переменных, каждая из которых характеризуется своим типом данных.

В Java различают примитивные (*primitive*) и ссылочные (*reference*) типы данных. К примитивным относятся:

– **целочисленные:** byte, short, int, long, char;
– **дробные:** float, double;
– **булевые:** boolean.

К ссылочным типам данных – классы, интерфейсы, массивы.

Длина числовых данных и границы допустимых значений представлены в табл. 1.1.

Таблица 1.1

Числовые типы данных

Название типа	Длина (байты)	Область значений
byte	1	от 128 до 127
short	2	от –32 768 до 32 767
int	4	от –2 147 483 648 до 2 147 483 647
long	8	от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 (примерно 10^{19})
char	2	'\u0000'–'\uffff', или 0–65 535
float	4	3,4e–38 < |x| < 3,4e38
double	8	1,7e–308 < |x| < 1,7e308

Все поля класса должны быть обязательно объявлены. При объявлении указывается имя и тип поля. Например:

```
int year;
```

При необходимости при объявлении можно указать начальное значение:

```
int year = 2011;
```

Методы

Метод (*method*) – фрагмент программного кода, задающий поведение объекта (экземпляра класса).

Метод состоит из заголовка и тела метода. Его структура имеет следующий вид:

```
тип имя_метода([список формальных параметров]) {
    команды метода
    возврат результата;
}
```

В заголовке метода записывается тип возвращаемого им значения (или ключевое слова void, если метод ничего не возвращает), имя метода и список формальных параметров в круглых скобках. Если формальные параметры отсутствуют, то после имени метода ставятся пустые круглые скобки.

Тело метода содержит команды, которые заключаются в фигурные скобки.

Формальные параметры в списке перечисляются через запятую. Для каждого сначала указывается тип, затем имя параметра. Например, метод для вычисления площади прямоугольника может выглядеть следующим образом:

```
int calculateArea(int width, int height) {
   return width * height;
}
```

В теле метода сначала вычисляется произведение `width * height`, а затем команда `return` возвращает полученный результат. Так как переменные `width` и `height` имеют целый тип, то и их произведение тоже будет целым, поэтому в заголовке метода записан тип `int`.

Метод:

```
void calculateArea(int width, int height) {
   System.out.print ("Площадь = " + (width * height));
}
```

вычисляет произведение `width * height`, но не возвращает результат, а выводит его на консоль, поэтому в заголовке метода вместо типа указано ключевое слово `void`.

В заголовке метода **для каждого формального параметра** обязательно должен быть указан свой тип. Например:

```
int calculateArea(int width, height)       // ошибка!
int calculateArea(int width, int height)   // правильная запись
```

Сигнатура метода (*signature*) определяется именем метода и его формальными параметрами (их количеством, типом, порядком следования). Нельзя создавать несколько методов с одинаковыми сигнатурами.

Конструктор

Книга – это абстрактное понятие. В реальном мире существуют конкретные экземпляры книг. Например: Монахов, В.В. Язык программирования Java и среда NetBeans, БХВ-Петербург, 2011. – 704 с. или Эккель, Б. Философия Java. – «Питер». – 2009. – 638 с.

Таким образом, Книга – это шаблон, описывающий любую книгу, а книга Монахова «Язык программирования Java …» и книга Эккеля «Философия Java» – экземпляры класса, обладающие конкретными характеристиками: названиями, годом издания и т. д.

Для создания экземпляров класса используются конструкторы.

Конструктор (*constructor*) – именованный фрагмент программного кода, предназначенный для инициализации экземпляра класса.

В общем виде команда создания экземпляра класса имеет следующий вид:

```
public Имя_класса имя_экземпляра_класса = new Имя_конструктора ([значения]);
```

Первым указывается модификатор доступа public. Он обозначает возможность работы с данным классом (то есть возможность создания и использования экземпляров этого класса) в другом классе (и даже в другом пакете).

Имя конструктора совпадает с именем класса. В классе может быть несколько конструкторов, которые отличаются типом или количеством параметров. Конструктор указывается после ключевого слова new при создании экземпляра класса.

Конструктор определяет действия, выполняемые при создании экземпляра класса, и является важной частью класса. Без конструктора невозможно создание экземпляров класса, поэтому если конструктор в классе не определен, Java предоставляет конструктор по умолчанию, который инициализирует экземпляр класса значениями по умолчанию. Например, при использовании конструктора по умолчанию:

```
public Book monakhov = new Book();
```

создается переменная monakhov. В памяти выделяется место для полей экземпляра класса Book. Переменной monakhov присваивается ссылка на созданный экземпляр класса Book, полям которого будут присвоены значения по умолчанию: полям bookTitle, author и publishingHouse – null (так как они являются ссылочными данными), а полям year и pageCount – 0 (так как они являются целыми числами).

Если же в классе определен конструктор с параметрами, то конструктор по умолчанию становится недоступным, и для его использования необходимо явное объявление такого конструктора.

В конструкторе, как и в методе, выделяют заголовок и тело конструктора. Подобно методу, у конструктора могут быть формальные параметры, которые передаются для инициализации объекта.

Например, конструктор класса Book может иметь следующую реализацию:

```
public Book(String bookTitle, String author, String publishing,
      int year, int pageCount) {
    this.bookTitle = bookTitle;
    this.author = author;
    this.publishing = publishing;
    this.year = year;
    this.pageCount = pageCount;
}
```

В заголовке конструктора в круглых скобках перечислены формальные параметры с указанием типа каждого из них.

В теле конструктора записаны команды, выполняющее присваивание значений формальных параметров полям класса. Для того чтобы не было конфликта при использовании одинаковых имен формальных параметров и полей класса, перед именем поля указывается ключевое слово this.

Внимание! Специальная ссылка на объект **this** используется для указания на текущий объект, для которого был вызван данный метод. Ссылка **this** чаще всего используется для передачи ссылки на текущий объект в качестве параметра метода.

Например, при создании экземпляра книги Эккеля «Философия Java» может использоваться вызов следующего конструктора:

```
public Book eckel = new Book("Философия Java", "Эккель Б.",
"Питер", 2009, 638);
```

В результате работы данного конструктора создается переменная eckel.

В памяти выделяется место для экземпляра класса Book. Переменной eckel присваивается ссылка на созданный экземпляр класса Book, полям которого будут присвоены значения формальных параметров конструктора: полям bookTitle – «Философия Java», author – «Эккель Б.», publishingHouse – «Питер», year – 2009, pageCount – 638.

Для обращения к полю экземпляра класса необходимо записать его имя, затем после точки указать нужное поле. Например:

```
eckel.bookTitle
```

Метод вызывается аналогично:

```
eckel.toString();
```

Итак, мы рассмотрели назначение и правила записи членов класса. Как же описывается класс в Java? Какова его структура?

Структура класса

Описание класса начинается с зарезервированных слов public и class, после которых записывается имя класса. Далее в фигурных скобках размещается тело класса, которое содержит объявление его полей, методов и конструкторов. Схематично класс можно представить следующим образом:

```
public class Имя_класса {
    Объявление полей
    Описание конструкторов
    Описание методов
}
```

Опишем класс «Книга»:

```
public class Book {
    // Объявление полей
    String bookTitle;          // название книги
```

```
    String author;          // автор
    String publishingHouse;     // издательство
    int year;               // год выпуска
    int pageCount;          // количество страниц

    // Конструктор класса
    public Book(String bookTitle, String author, String publishing,
        int year, int pageCount) {
        this.bookTitle = bookTitle;
        this.author = author;
        this.publishing = publishing;
        this.year = year;
        this.pageCount = pageCount;
    }

    // Метод
    double calculateCountPrintedSheets(int pageCount){
        return pageCount / 8 * 0.93;
    }

}
```

Обратите внимание на то, что описание полей, конструктора и метода сдвинуто влево относительно заголовка класса на позицию табуляции, а в каждой строке с описанием полей присутствуют комментарии.

Наличие комментариев в программном коде – признак хорошего тона. Комментарии не влияют на ход выполнения программы, так как компилятор пропускает их при компиляции программы, а правильно записанные комментарии улучшают читаемость кода и понимание логики программы.

Комментарии к классу, полям, методам, конструктору

В языке программирования Java различают строчные, блочные комментарии и комментарии документации.

Строчные комментарии обозначаются двумя наклонными чертами и продолжаются до конца строки. Их используют, как правило, для пояснения назначения полей.

В комментариях к классам и методам можно использовать блочные комментарии или комментарии документации.

Блочные комментарии содержат несколько строк, которые заключаются между символами `/* ..... */`.

Комментарии документации предназначены для автоматизации создания документации к приложению с помощью утилиты javadoc. Они оформляются аналогично блочным, но начинаются с символов /**, а каждая внутренняя строка начинается с символа *. Комментарии документации могут содержать специальные теги, которые начинаются с символа @. Подробное описание этих тегов приводится в прил. 2.

В комментарии к классу обычно помещают строки с описанием назначения класса, фамилией автора и версией программного кода. Эти строки содержат теги @author и @version.

Комментарии к методу могут быть как строчными, так и блочными. Если метод имеет список формальных параметров или возвращаемое значение, то их можно записать с помощью специальных тегов @param и @return, описание которых также приводится в прил. 2.

Класс «Книга» с комментариями будет выглядеть следующим образом:

```
/**
 * Класс, описывающий книгу
 * @author Болбот О.М.
 * @version 1.0
 */
public class Book {
    String bookTitle;           // название книги
    String author;         // автор
    String publishingHouse;     // издательство
    int year;              // год выпуска
    int pageCount;         // количество страниц

    /**
     * Метод для вычисления количества печатных листов
     * @param pageCount - количество страниц формата А4
     * @return double - количество печатных листов
     */
    double calculateCountPrintedSheets(int pageCount){
        return pageCount / 8 * 0.93;
     }

}
```

Компиляция и запуск

Программный код класса можно набрать в любом текстовом редакторе и сохранить в текстовом файле с именем Book.java. Затем можно вызвать компилятор и передать ему имя этого файла в качестве аргумента:

```
javac Book.java
```

Из текстового файла компилятор создаст файл с байт-кодами, даст ему имя Book.class и запишет этот файл в ту же папку.

Для выполнения нужно вызвать интерпретатор байт-кодов, передав ему имя файла с байт-кодами (имя файла записывается без расширения) в качестве аргумента:

```
java Book
```

Для автоматизации работы по созданию приложения можно использовать различные интегрированные среды разработки Java-приложений: NetBeans, Eclipse, 7.4 JDeveloper, JBuilder и др. Далее в пособии будет рассматриваться среда NetBeans, основные сведения, необходимые для работы, приводятся в прил. 1.

Постановка задачи

Работу с классами рассмотрим на примере следующей задачи. Представим себе работу офиса издательства, выпускающего печатную продукцию: книги, учебники, журналы и т. п.

Проанализировав данную предметную область, можно выделить такие сущности, как авторы, публикуемые материалы, издаваемая продукция, работники издательства. Необходимо создать проект и разработать классы, которые будут моделировать эти сущности и взаимодействие между ними.

Назовем проект «Издательство». Он будет содержать классы «Автор», «Публикуемый материал», «Книга», «Журнал», а также собственно класс «Издательство». Для каждого из них необходимо определить и разработать структуру: поля, методы и конструкторы.

Среди классов проекта по своему функциональному назначению выделяется класс «Издательство». Назовем его управляющим классом.

Задание 1. Создайте новый проект «Издательство» (publishing).

Порядок работы

1. Запустите NetBeans.
2. Нажмите кнопку на панели инструментов или откройте меню Файл → Создать проект.
3. Выберите категорию проекта: Java; тип проекта: Приложение Java (рис. 1.1) и нажмите кнопку **Далее >**.

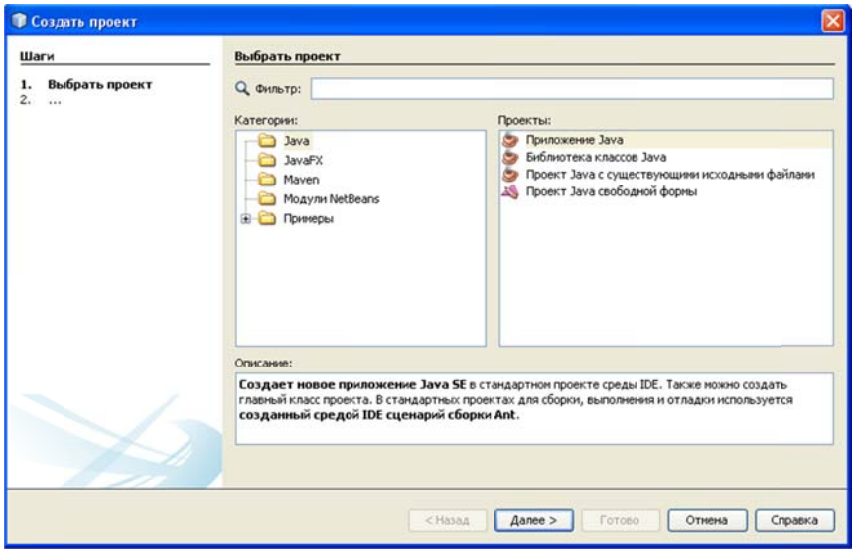

Рис. 1.1. Первый шаг создания проекта

4. Имена проектов принято записывать словами на английском языке. Введите имя проекта: publishing, выберите место его расположения и нажмите кнопку **Готово** (рис. 1.2).

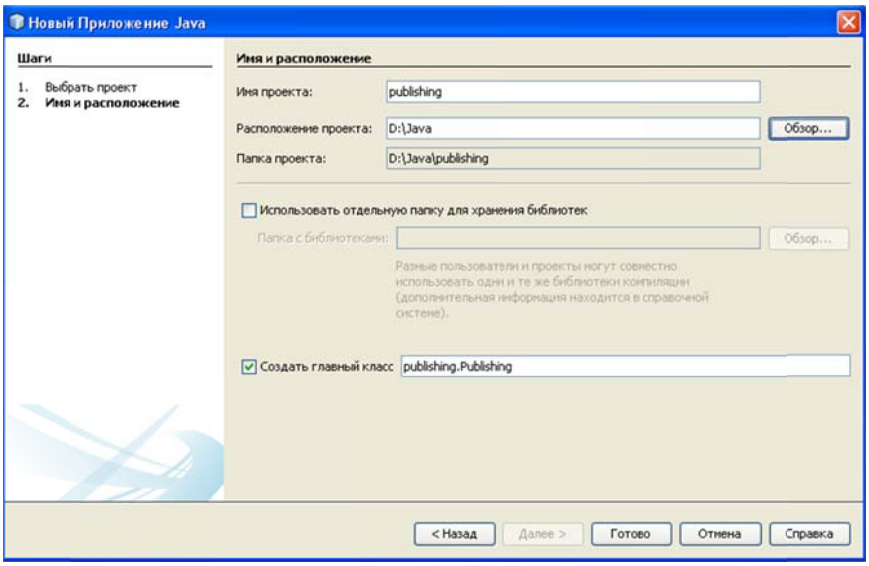

Рис. 1.2. Второй шаг создания проекта

5. Удалите комментарии, расположенные выше заголовка проекта.

6. Запишите комментарии к классу.

7. Удалите комментарии, размещенные в методе main. В результате должен получиться программный код, изображенный на рис. 1.3.

```
1   package publishing;
2
3   /**
4    * Класс "Издательство"
5    * @author Болбот О.М.
6    */
7   public class Publishing {
8
9       public static void main(String[] args) {
10
11      }
12
13  }
```

Рис. 1.3. Программный код класса Publishing

Задание 2. В проекте publishing создайте новый класс, описывающий автора и содержащий 5–6 полей. Назовите класс и его поля согласно соглашениям кодирования Java. Определите типы данных. Создайте конструктор. Создайте метод toString для вывода полной информации об экземпляре класса и метод, возвращающий фамилию и инициалы автора.

В соответствии с соглашением о кодировании имя класса должно быть английским существительным, записанным с прописной буквы. Назовем его: Author. Сведения, которые нужны издательству для работы с авторами: фамилия, имя, отчество, паспортные данные, адрес, контактный телефон. Имена полей класса должны быть английскими существительными, записанными строчными буквами.

Имена и типы полей класса Author представлены в табл. 1.2.

Таблица 1.2

Имена и типы полей класса Author

Имя поля	Тип поля	Описание
surname	String	Фамилия
name	String	Имя
secondName	String	Отчество
passportData	String	Паспортные данные
address	String	Адрес
phoneNumber	String	Контактный телефон

Порядок работы

1. Для создания класса Author нажмите кнопку  на панели инструментов или откройте меню Файл → Создать файл.

2. Выберите Категорию: Java; Тип файла: Класс Java и нажмите кнопку **Далее >** (рис. 1.4).

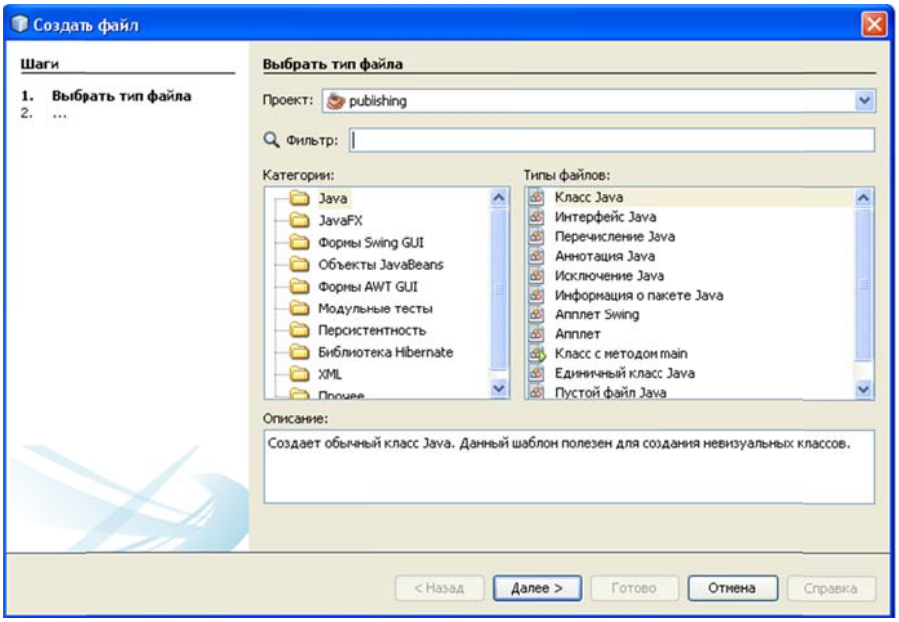

Рис. 1.4. Первый шаг создания класса

3. Введите имя класса: Author и нажмите кнопку **Готово** (рис. 1.5).

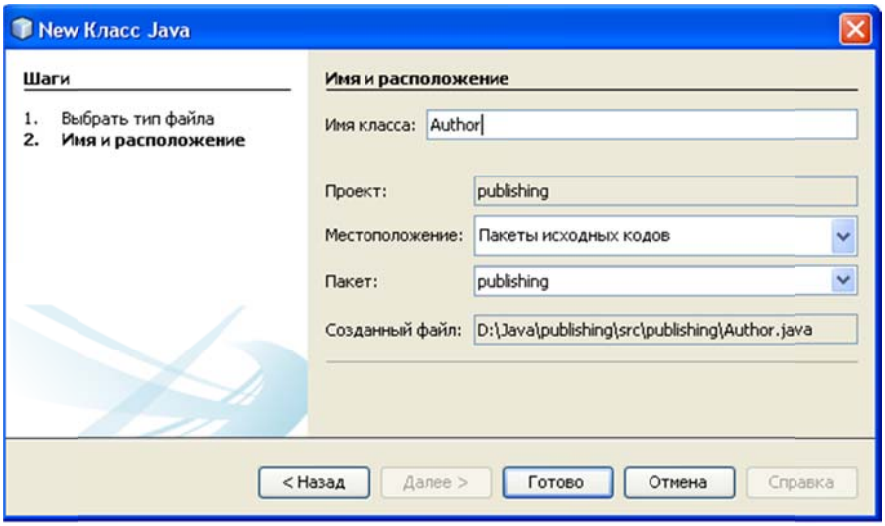

Рис. 1.5. Второй шаг создания класса

4. Удалите комментарии, расположенные выше заголовка проекта.
5. В комментариях к классу укажите автора и запишите назначение класса.
6. После заголовка класса объявите его поля.
7. В результате получится код, изображенный на рис. 1.6.

```
1   package publishing;
2
3   /**
4    * Класс, описывающий автора публикуемого материала
5    * @author Болбот О.М.
6    */
7   public class Author {
8       String surname;          // фамилия
9       String name;             // имя
10      String secondName;       // отчество
11      String passportData;     // паспортные данные
12      String address;          // адрес
13      String phoneNumber;      // контактный телефон
14
15  }
```

Рис. 1.6. Программный код класса Author

8. После объявления полей класса Author добавьте код конструктора:

```
/**
 * Конструктор для создания экземпляра автора
 * @param surname - фамилия
 * @param name - имя
 * @param secondName - отчество
 * @param passportData - паспортные данные
```

```
     * @param address - адрес
     * @param phoneNumber - контактный телефон
     */
    public Author(String surname, String name, String secondName,
           String passportData, String address, String phoneNumber) {
        this.surname = surname;
        this.name = name;
        this.secondName = secondName;
        this.passportData = passportData;
        this.address = address;
        this.phoneNumber = phoneNumber;
    }
```

9. Создайте метод toString.

Данный метод используется для формирования символьной строки, содержащей информацию об экземпляре класса. Программный код метода toString для класса Author приводится ниже (аннотация @Override используется для переопределения метода).

```
    /**
     * Метод, формирующий полную информацию об авторе
     * @return the str - возвращает строку
     */
    @Override
    public String toString() {
       String str = surname + " " + name + " " + secondName + ", " +
              "паспортные данные: " + passportData + ", " +
              "\nадрес: " + address + ", телефон: " + phoneNumber;
       return str;
    }
```

10. Создайте метод, возвращающий фамилию и инициалы автора. Формальными параметрами метода будут выступать фамилия, имя и отчество. Для выбора первой буквы имени и отчества используется метод substring:

```
    /**
     * Метод, возвращающий фамилию и инициалы автора
     * @param surname - фамилия
     * @param name - имя
     * @param secondName - отчество
     * @return the str - возвращает строку типа Иванов И.И.
     */
    String getShortName(String surname, String name, String second-
Name) {
        String str = surname + " " + name.substring(0, 1) + "." +
               secondName.substring(0, 1) + ".";
        return str;
    }
```

11. Сохраните работу, открыв меню Файл → Сохранить или нажав кнопку  на панели инструментов.

12. Создайте экземпляр класса Author и выведите на консоль полную информацию о нем, затем – только фамилию и инициалы. Для этого перейдите на вкладку класса Publishing и в методе main запишите следующие команды:

```
    public static void main(String[] args) {
        // Создание автора
        System.out.println("Авторы:");
        Author vasilev = new Author("Васильев", "Павел", "Иванович",
                    "MP2035648, выдан 18.05.2011",
                    "г.Минск, ул.Васнецова, д.45, кв.79",
                    "8(029)33-564-78-02");
        System.out.println(vasilev.toString());
        String shortNameVasilev = vasilev.getShortName(vasilev.get
Surname(),
                    vasilev.getName(), vasilev.getSecondName());
        System.out.println("Фамилия и инициалы: " + shortNameVasilev);
    }
```

13. Если выполнить проект, то на консоли появятся сообщения:

```
run:
Авторы:
Васильев Павел Иванович, паспортные данные: MP2035648, выдан
18.05.2011,
адрес: г.Минск, ул.Васнецова, д.45, кв.79, конт.телефон:
8(029)33-564-78-02
Фамилия и инициалы: Васильев П.И.
СБОРКА УСПЕШНО ЗАВЕРШЕНА (общее время: 0 секунд)
```

Статические члены класса

Разные экземпляры одного и того же класса имеют поля, принимающие различные значения. Изменение поля одного экземпляра не влияет на значение того же поля в другом экземпляре, потому что в каждом экземпляре для таких полей выделяется своя ячейка памяти. Такие поля называются переменными экземпляра класса (*instance variables*).

Иногда бывает необходимо создать поле, общее для всего класса, не принадлежащее отдельному экземпляру класса. Например, в издательстве необходимо вести учет поступивших материалов, то есть в классе «Публикуемый материал» необходимо создать поле, которое будет содержать порядковый номер последнего поступившего материала. Такие поля называются статическими, или переменными класса (*class variables*). Для переменных класса выделяется только одна ячейка памяти общая для всех экземпляров класса, независимо от того, сколько их было создано.

Переменные класса помечаются в языке программирования Java модификатором static. Этот модификатор используется в том случае, когда нужно создать такое поле, которое относилось бы не к экземпляру класса, а собственно к классу. Это означает создание переменной, которая должна изменяться под воздействием класса, а не объекта.

Статическими могут быть как поля, так и методы. Для методов модификатор static используются так же, как и для полей класса. Модификатор static говорит о том, что данный метод может вызываться только посредством класса, а не экземпляра класса. Необходимо помнить, что статические методы могут осуществлять доступ только к статическим переменным.

Чтобы вызвать статический метод или использовать статическое поле, нужно написать имя класса, затем поставить точку, а затем записать имя метода (если необходимо его вызвать) или имя поля (если необходимо получить его значение или записать в него новое значение):

```
имя_класса.статический_метод()
имя_класса.статическое_поле
```

Задание 3. В проекте **publishing** создайте новый класс для описания публикуемого материала. Опишите поля класса. Одним из полей должен быть порядковый номер публикуемого материала. Создайте метод toString для вывода информации об экземпляре класса и метод для подсчета общего количества поступивших в издательство материалов.

Имя класса должно быть английским существительным, записанным с прописной буквы: Note. Определим поля, которые должны быть у любой публикации: порядковый номер, название материала, фамилии и инициалы автора, объем материала (в авторских листах).

Для формирования порядковых номеров, присваиваемых поступающим в издательство материалов, необходимо объявить еще одно поле, в котором будет храниться последний номер поступившего в издательство материала. Оно будет статическим. При объявлении присвоим ему нулевое значение.

Имена полей класса должны быть английскими существительными, записанными строчными буквами. Имена и типы полей класса Note представлены в табл. 1.3.

Таблица 1.3

Имена и типы полей класса Note

Имя поля	Тип поля	Описание
numberNote	int	Порядковый номер материала
lastNumberNote	int статическое поле	Последний порядковый номер
noteTitle	String	Название материала
authorNote	String	Автор материала
numberAuthorsSheets	double	Количество авторских листов

Порядок работы

1. В проекте publishing создайте новый класс Note.
2. Удалите комментарии, расположенные выше заголовка проекта.
3. В комментариях к классу запишите автора и назначение класса.
4. После заголовка класса объявите поля класса:

```
int numberNote;                 // порядковый номер материала
static int lastNumberNote = 0;  // последний порядковый номер
String noteTitle;               // название материала
String authorNote;              // автор материала
double numberAuthorsSheets;     // количество авторских листов
```

5. Создайте конструктор класса Note. Переменная numberNote экземпляра класса будет получать порядковый номер из статической переменной класса lastNumberNote, поэтому ее не нужно указывать в списке формальных параметров конструктора.

```
/**
 * Конструктор для создания экземпляра публикуемого материала
 * @param noteTitle - название материала
 * @param authorNote - автор материала
 * @param numberAuthorsSheets - количество авторских листов
 */
public Note(String noteTitle, String authorNote,
            double numberAuthorsSheets) {
     numberNote = ++lastNumberNote; // создаем порядковый номер
     this.noteTitle = noteTitle;
     this.authorNote = authorNote;
     this.numberAuthorsSheets = numberAuthorsSheets;
}
```

6. Создайте метод toString:

```
/**
 * Метод, формирующий информацию о публикуемом материале
 * @return the str - возвращает строку
 */
@Override
public String toString() {
     String str = numberNote + " " + authorNote + " " + noteTitle +
               ", авт. листов: " + numberAuthorsSheets;
     return str;
}
```

7. Метод для подсчета количества поступивших в издательство материалов является статическим и будет возвращать значение переменной класса lastNumberNote:

```
    /**
     * Метод для подсчета количества поступивших материалов
     * @return the lastNumberNote
     */
    static int getLastNumberNote() {
        return lastNumberNote;
    }
```

8. Создайте метод, вычисляющий количество страниц формата А4:

```
    /**
     * Метод подсчета количества страниц формата А4
     * @return int - кол-во страниц формата А4
     * Из поля numberAuthorsSheets получает количество авторских
листов;
     * вычисляет по формуле количество страниц формата А4;
     * с помощью метода round класса Math округляет полученный ре-
зультат
     * и возвращает его в виде целого числа.
    */
    public int getNumberPages() {
            return (int) Math.round(numberAuthorsSheets * 40000 /
3700);
    }
```

9. Сохраните работу, открыв меню Файл → Сохранить или нажав кнопку  на панели инструментов.

10. Для проверки правильности вычисления порядкового номера публикуемого материала создайте два экземпляра класса Note и выведите на консоль информацию о них. Для этого перейдите на вкладку класса Publishing и в методе main ниже запишите следующие команды:

```
    // Создание материала
    System.out.println("\nПубликуемые материалы:");
    Note noteVasilev1 = new Note("Создание классов в Java",
            shortNameVasilev, 3.6);
    Note noteVasilev2 = new Note("Среда NetBeans",
            shortNameVasilev, 2.8);
    System.out.println(noteVasilev1.toString());
    System.out.println("Кол-во страниц " + noteVa-
silev1.getNumberPages());
    System.out.println(noteVasilev2.toString());
    System.out.println("Кол-во страниц " + noteVa-
silev2.getNumberPages());
    System.out.println("\nКоличество публикуемых материалов:" +
            Note.getLastNumberNote());
```

11. Выполните проект. На консоль будет выведена информация:

```
run:
Авторы:
Васильев Павел Иванович, паспортные данные: MP2035648, выдан
18.05.2011,
адрес: г.Минск, ул.Васнецова, д.45, кв.79, конт.телефон:
8(029)33-564-78-02
Фамилия и инициалы: Васильев П.И.

Публикуемые материалы:
1 Васильев П.И. Создание классов в Java, авт. листов: 3.6
Кол-во страниц 39
2 Васильев П.И. Среда NetBeans, авт. листов: 2.8
Кол-во страниц 30

Количество публикуемых материалов: 2
```

Краткие итоги

В Java имеются встроенные примитивные типы: целые byte, short, int, long, символьный тип char, вещественные типы float и double и тип boolean; классы, интерфейсы, массивы – ссылочные.

При выборе имен классов, полей и методов необходимо руководствоваться соглашениями кодирования Java.

Класс – это описание того, как устроен объект. Класс содержит поля, методы и конструкторы. Методы задают поведение, а поля – состояние объекта.

Экземпляр класса создается с помощью команды new и обращения к нужному конструктору. При этом для экземпляра класса выделяется память, а полям присваиваются конкретные значения.

Объявление метода состоит из заголовка и тела метода (его реализации).

Параметры, указанные в заголовке метода при его описании, – формальные, а подставляемые во время вызова – фактические. Формальные параметры нужны для указания последовательности задания фактических параметров во время вызова метода.

Сигнатура метода определяется именем метода и его формальными параметрами (их количеством, типом, порядком следования). В одном классе не должно быть двух методов с одинаковыми сигнатурами.

Различают переменные и методы экземпляров класса, переменные и методы класса. Последние являются статическими и помечаются модификатором static.

Программный код необходимо комментировать. Если использовать комментарии документации, то можно с помощью утилиты javadoc оформить документацию к приложению в формате HTML.

Для автоматизации работы с программным кодом можно использовать интегрированные среды разработки (*Integrated Development Environment* – IDE), одной из которых является среда NetBeans.

2. ИНКАПСУЛЯЦИЯ. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Принцип инкапсуляции

Инкапсуляция (*encapsulation*) – это сокрытие реализации класса и отделение его внутреннего представления от внешнего. При использовании объектно-ориентированного подхода не принято применять прямой доступ к полям какого-либо класса из методов других классов. Для получения и изменения значения полей класса используются специальные методы.

Внутри класса данные и методы могут обладать различной степенью открытости (или доступности). Открытые члены класса составляют внешний интерфейс объекта. Это та функциональность, которая доступна другим классам. Закрытыми обычно объявляются все поля класса, а также вспомогательные методы, которые являются деталями реализации, от них не должно зависеть взаимодействие с другими классами.

Благодаря сокрытию реализации, можно менять внутреннюю логику отдельного класса, не меняя код остальных компонентов системы.

Для этого с помощью модификатора доступа private необходимо сделать члены класса недоступными извне, а для доступа к отдельным полям класса использовать методы get и set. Метод get используется для получения, set – для изменения значений полей экземпляра класса.

Обеспечение доступа к полям класса только через его методы дает ряд преимуществ: гораздо проще контролировать значения полей, так как при прямом обращении к свойствам им могут присвоить некорректные значения. Программный код, написанный с использованием данного принципа, легче отлаживать.

Модификаторы доступа

Модификатором называется зарезервированное слово, задающее правила доступа к элементу или особенность его реализации. Его ставят перед типом и именем задаваемого поля, метода или класса. Разрешается писать подряд несколько модификаторов, поясняющих разные особенности элемента.

Классы, поля и методы могут иметь модификаторы (спецификаторы) доступа, которые определяют область их доступности. В Java имеются следующие модификаторы доступа (в порядке увеличения области доступности):

- private (закрытый) – доступ только внутри класса;
- protected (защищенный) – доступ внутри пакета и в дочерних классах;
- public (открытый) – доступ внутри и вне класса и пакета.

Все, что объявлено модификатором private, доступно только внутри класса. Поля или методы, объявленные модификатором public, доступны из любого места вне класса. Если модификатор доступа не указан, то область видимости ограничивается пакетом. Области видимости для модификаторов доступа представлены в табл. 2.1.

Таблица 2.1

Модификаторы и зоны доступа

Модификатор	Тело класса	Пакет, содержащий класс	Класс-наследник (подкласс)	Вся остальная часть программы (например, другие пакеты)
public	+	+	+	+
protected	+	+	+	–
default (модификатор не пишется)	+	+	–	–
private	+	–	–	–

Из таблицы следует, что модификатор public предоставляет доступ из любой части программы. Это самый открытый и общедоступный вариант. Напротив, модификатор private разрешает обращаться напрямую к члену класса только из самого класса. Это самый закрытый вариант.

Методы get и set

Использование модификатора доступа private делает поля класса недоступными извне. Если же необходимо получить доступ к отдельным полям класса, то обычно используют методы get и set. Метод get используется для получения, а метод set – для изменения значения полей экземпляра класса. Кроме того, в методе set может корректироваться неверно заданное значение. Эти методы называются «геттеры» (от слова *get* – получать) и «сеттеры» (от слова *set* – устанавливать).

Таким образом, инкапсуляция позволяет контролировать использование членов класса за его пределами.

Задание 4. Реализуйте возможность «прямого» изменения значения полей экземпляра одного класса из другого: например, из класса Publishing измените информацию о книге «Создание классов в Java».

Порядок работы

1. Перейдите на вкладку класса Publishing.
2. В метод main добавьте команды изменения автора и порядкового номера книги «Создание классов в Java»:

```
noteVasilev1.numberNote = 20;
noteVasilev1.authorNote = "Петров Д.В.";
System.out.println(noteVasilev1.toString());
```

3. Выполните проект и просмотрите результат. На консоль будут выведены изменения:

```
Публикуемые материалы:
1 Васильев П.И. Создание классов в Java, авт. листов: 3.6
Кол-во страниц 39
2 Васильев П.И. Среда NetBeans, авт. листов: 2.8
Кол-во страниц 30
20 Петров Д.В. Создание классов в Java, авт. листов: 3.6
```

4. Таким образом, достоверность информации о книге Васильева «Создание классов в Java» нарушена. Это стало возможным, потому что при описании полей класса Note не было указано никакого модификатора доступа. В этом случае действует правило: по умолчанию поле имеет модификатор доступа default, то есть поле не защищено от изменения и доступно для модификации в любом классе в пределах пакета. Чтобы случайные изменения значения полей стали невозможны, необходимо запретить прямой доступ из других классов к полям класса Note.

Задание 5. Инкапсулируйте поля класса Note.

Для инкапсуляции полей необходимо в объявлении полей класса записать модификатор доступа private. Для получения значений полей экземпляров данного класса необходимо для всех полей создать методы get***. Исходя из функционального назначения полей класса Note, новые значения можно устанавливать только для полей «Название» (возможно, автор захочет его изменить) и «Объем материала» (при редактировании материала может измениться его объем). Поэтому методов set*** в данном классе будет меньше, чем методов get***, – только два: setNoteTitle и setNumberAuthorsSheets.

Порядок работы

1. Перейдите на вкладку класса Note.
2. В командах объявления полей класса добавьте модификатор private:

```
private int numberNote; // порядковый номер материала
private static int lastNumberNote = 0; // последний номер
private String noteTitle; // название материала
private String authorNote; // автор материала
private double numberAuthorsSheets; // кол-во авт. листов
```

3. Сохраните изменения.
4. Проверьте невозможность изменения значений полей экземпляра класса Note из другого класса. Для этого перейдите в класс Publishing. В командах метода main (в которых изменялись порядковый номер и автор книги «Создание классов в Java») среда NetBeans выделила волнистой красной линией ошибки. При наведении указателя мыши появляется всплывающая подсказка: numberNote has private access in Note (доступ к полю numberNote – только

в классе Note). Таким образом, из другого класса стало невозможно изменить поля класса Note. Удалите команды с ошибками.

5. Теперь поля класса Note стали недоступны из других классов, и для получения значений полей экземпляра класса Note извне необходимо создать методы get_Имя_поля, для установки новых значений – методы set_Имя_поля.

6. Для всех полей класса создайте методы для чтения их значений по аналогии с приведенным ниже методом:

```
/**
 * Метод чтения автора материала
 * @return authorNote
 */
public String getAuthorNote() {
     return authorNote;
}
```

7. В заголовок метода getLastNumberNote также добавьте модификатор доступа public:

```
public static int getLastNumberNote() {
     return lastNumberNote;
}
```

8. Создайте методы изменения значений для полей noteTitle и numberAuthorsSheets:

```
/**
 * Метод изменения названия материала
 * @param noteTitle - новое название
 */
public void setNoteTitle(String noteTitle) {
     this.noteTitle = noteTitle;
}

/**
 * Метод изменения количества авторских листов
 * @param numberAuthorsSheets - новое количество авторских ли-
стов
 */
public void setNumberAuthorsSheets(double numberAuthorsSheets)
{
     this.numberAuthorsSheets = numberAuthorsSheets;
}
```

9. Проверьте работу созданных методов. Для этого перейдите в класс Publishing. В метод main добавьте команды изменения названия книги «Создание классов в Java» и количества авторских листов:

```
noteVasilev1.setNoteTitle("Классы в Java");
noteVasilev1.setNumberAuthorsSheets(3.5);
System.out.println(noteVasilev1.toString());
```

10. Выполните проект и просмотрите результат выполнения.

Задание 6. Инкапсулируйте поля класса Author.

Инкапсуляцию полей можно выполнить, используя возможности среды NetBeans.

Порядок работы

1. Перейдите на вкладку класса Author.
2. Откройте меню Реорганизация кода и выберите команду Инкапсулировать поля (Refactoring → Incapsulate fields).
3. В открывшемся окне Инкапсулировать поля установите флажки для создания всех методов получения значений полей класса (методов get***). Методы установки необходимо создавать только для тех полей, которые могут изменяться: например, для изменения адреса, телефона автора и т. п. (рис. 2.1).

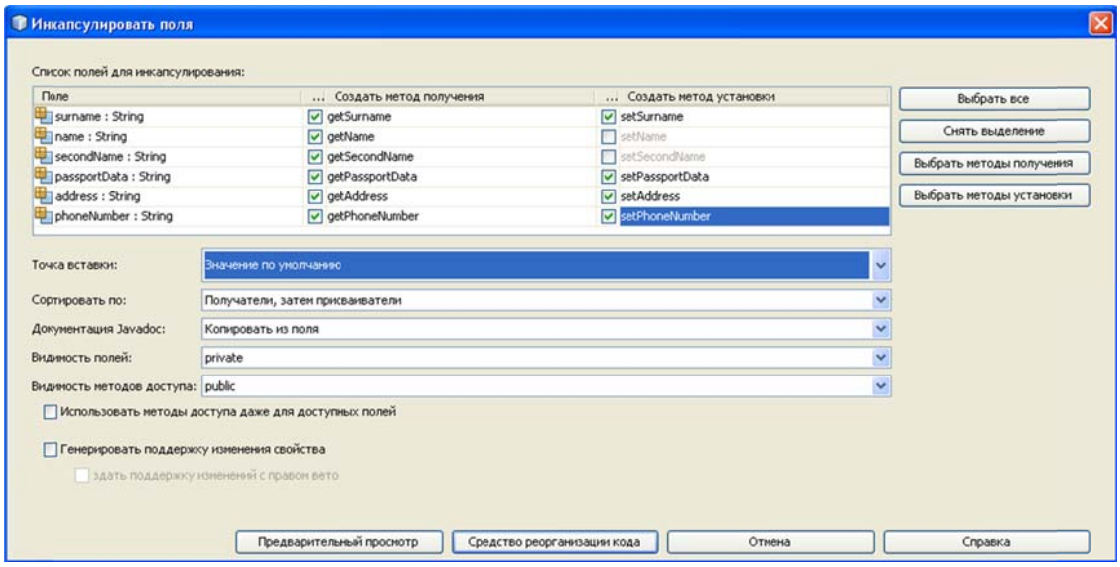

Рис. 2.1. Инкапсуляция полей в среде NetBeans

4. В списке Видимость полей выберите: private, в списке – Видимость методов доступа public.
5. После этого нажмите кнопку Средство реорганизации кода. Просмотрите полученный код.
6. Для созданных методов чтения и установки полей класса оформите комментарии к методам согласно правилам создания комментариев документации Java: опишите назначение метода, формальные параметры метода – с помощью тега @param, возвращаемое значение с помощью тега @return.
11. Для проверки работы методов измените какую-нибудь информацию об авторе Васильеве: например, номер телефона. Для этого перейдите в класс Publishing. В метод main добавьте:

```
vasilev.setPhoneNumber("8(029)44-564-78-02");
System.out.println(vasilev.toString());
```

12. Самостоятельно измените адрес и паспортные данные Васильева.
13. Выполните проект и просмотрите результат выполнения.

Краткие итоги

Область видимости полей и методов класса указывается с помощью модификаторов доступа. В Java имеются следующие модификаторы доступа:

– private (закрытый) – доступ только внутри класса;
– protected (защищенный) – доступ внутри пакета и в дочерних классах;
– public (открытый) – доступ внутри и вне класса и пакета.

По умолчанию область видимости ограничивается пакетом.

Общая форма объявления полей выглядит следующим образом:

```
[модификаторы] тип имя_поля [= значение];
```

В квадратных скобках записаны необязательные элементы объявления.

Объявление метода состоит из заголовка и тела метода. Заголовок состоит:

– из модификаторов;
– типа возвращаемого значения или ключевого слова void (если метод ничего не возвращает);
– имени метода;
– списка формальных параметров в круглых скобках (их может не быть).

Общая форма объявления метода выглядит следующим образом:

```
[модификаторы] тип имя_метода([список_формальных_параметров]) {
      // тело метода
}
```

Структура класса может иметь следующий вид:

```
[модификатор доступа] class Имя_класса {
    [модификаторы] тип имя_поля_1;
    [модификаторы] тип имя_поля_2;

 [Модификатор доступа] Имя_конструктора(список_форм_парам-в){
            // тело конструктора
 }

 [Модификаторы] тип имя_метода(список_формальных_параметров){
      // тело метода
 }
}
```

Инкапсуляция – это сокрытие реализации класса. Для этого поля класса объявляются с модификатором private, а для доступа к ним используются методы get и set. Метод get используется для получения, а метод set – для изменения значений полей экземпляра класса.

3. UML-ДИАГРАММЫ КЛАССОВ

Унифицированный язык моделирования

UML – унифицированный язык моделирования (*Unified Model Language*) − графический язык визуализации, специфицирования, конструирования и документирования программного обеспечения. С помощью UML можно разработать проект приложения.

Идея использования UML-диаграмм заключается в том, что проект разрабатывается командой проектировщиков настолько детально, что он будет понятен для программистов, которые будут писать программный код.

Основными элементами UML являются сущности (*thing*), отношения (*relationship*), диаграммы (*diagram*). Сущности являются ключевыми абстракциями языка, отношения связывают сущности, диаграммы группируют коллекции сущностей.

Диаграмма классов (*Class diagram*)

Класс (*Class*) – описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. Графически класс отображается в виде прямоугольника, обычно включающего секции с именем, свойствами (атрибутами) и операциями.

У каждого класса должно быть имя, отличающее его от других классов.

Атрибут – это именованное свойство (поле) класса, общее для всех экземпляров этого класса. Класс может содержать любое число атрибутов или не содержать их вовсе. При описании атрибута можно явным образом указывать его класс, начальное значение и область видимости.

В языке UML можно определить три уровня видимости:

– public (открытый) – любой внешний класс, может пользоваться открытыми свойствами. Обозначается знаком + (плюс) перед именем атрибута или операции;

– protected (защищенный) – любой потомок данного класса может пользоваться его защищенными свойствами. Обозначается знаком # (диез);

– private (закрытый) – только данный класс может пользоваться закрытыми свойствами. Обозначается символом – (минус).

Операция – это абстракция того, что позволено делать с объектом. У всех объектов класса имеется общий набор операций. Класс может содержать любое число операций или не содержать их вовсе. Операции класса изображаются в разделе, расположенном ниже раздела с атрибутами.

В языке UML существует различие между операцией и методом. Операциями называются услуги, которые можно запросить у любого объекта класса для изменения поведения; метод – это реализация операции. Для одной операции может быть определено несколько методов, из которых нужный полиморфно выбирается во время выполнения.

На самом высоком уровне абстракции при создании UML-диаграммы классов можно просто записывать имена атрибутов и операций. Этого обычно бывает достаточно для понимания общего назначения модели. При более подробной детализации можно определить видимость и область действия атрибутов и операций. Для операций можно также задать параметры, тип возвращаемого значения и некоторые другие свойства.

Ниже приводятся примеры некоторых допустимых объявлений операций:

– display() – только имя;
– + display() – видимость и имя;
– set (n: Name, s: String) – имя и параметры;
– getID() : int – имя и возвращаемое значение.

На рис. 3.1 изображена диаграмма класса Note.

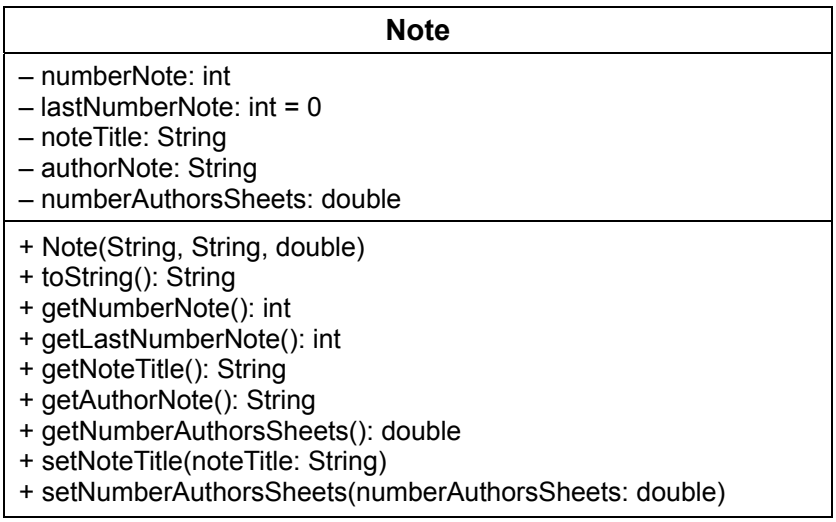

Рис. 3.1. UML-диаграмма класса Note

Диаграммой классов (*Class diagram*) называют диаграмму, на которой показано множество классов, интерфейсов и отношения между ними.

Диаграмма классов является основой для написания программы. Диаграмма описывает типы объектов и различные виды статических отношений, которые существуют между ними.

Задание 7. Разработайте UML-диаграмму класса Author.

Краткие итоги

UML – унифицированный язык моделирования, предназначенный для специфицирования, конструирования и документирования программного обеспечения. С его помощью можно разрабатывать различные диаграммы.

Классы проектируются с помощью UML-диаграмм. Диаграмма класса представляет собой прямоугольник, состоящий из трех секций: имя класса, атрибуты (поля) и операции (методы).

Диаграмма классов отображает множество классов и отношения между ними.

4. НАСЛЕДОВАНИЕ. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

В объектно-ориентированном программировании принято использовать имеющиеся классы в качестве заготовок для создания новых классов, которые на них похожи, но обладают более сложной структурой и/или отличаются поведением. Такую технологию разработки классов называют наследованием.

Наследование (*inheritance*) – это отношение между классами, при котором один класс использует структуру или поведение другого класса. Подклассы (дочерние классы) обычно дополняют или переопределяют унаследованную структуру и поведение родительского класса.

Часто класс-родитель называют суперклассом (*superclass*), а класс-наследник – подклассом (*subclass*).

Использование наследования способствует уменьшению повторения кода, созданного для описания схожих сущностей, а также способствует написанию более эффективного кода.

Набор классов, связанных отношением наследования, называется иерархией классов. Класс, стоящий во главе иерархии, называется базовым классом иерархии. В Java все классы являются потомками класса Object (он является базовым для всех классов).

Чем ближе к основанию иерархии находится класс, тем более общим и универсальным он является. Базовый класс всегда называют именем, которое характеризует все экземпляры классов-наследников и выражает наиболее общую абстракцию, применимую к ним.

Анализируя сущности какой-либо предметной области, выделяют общие характеристики. На их основе создается базовый класс, от которого наследуются дочерние классы, которые будут иметь как черты «родителя», так и свои. Получается, что класс-наследник – это специализированная версия родительского класса, которая наследует члены класса-родителя и добавляет свои собственные.

Для каждого класса-наследника можно указывать только один родительский класс. При этом никакой класс не может быть собственным «родителем».

Для описания наследования в строке объявления дочернего класса используется служебное слово `extends`, после которого указываете имя родительского класса. Дочерний класс наследует поля и методы родительского класса, объявленные с помощью модификаторов доступа public и protected. При наследовании private-члены класса доступны только тому классу, в котором были объявлены.

Таким образом, структура дочернего класса имеет следующий формат:

```
Модификаторы class Имя_Дочернего_Класса extends Имя_Класса_Родителя {
        объявление полей;
        объявление конструкторов;
        объявление методов;
    }
```

В фигурных скобках записывается реализация класса: описание его полей, методов и конструкторов. При этом члены класса, имеющиеся в родительском классе, снова описывать не нужно – они наследуются.

Переопределение методов

При необходимости в классе-наследнике реализация родительского метода может быть переопределена. Для этого используется аннотация @Override.

Переопределение метода (*overriding*) означает, что реализация метода, взятая из родительского класса, заменяется другой. Сигнатуры методов при этом должны быть идентичными. Переопределению подлежат только нестатические методы.

В приведенных ранее примерах было использовано переопределение методов при разработке классов Note и Author – метод toString, который является метод класса Object. Его назначение – выдавать текстовую строку с информацией об экземпляре класса. Поскольку классы обладают различными полями, то метод необходимо переопределять для каждого класса.

Члены класса с модификатором final

Если в классе присутствуют его члены, значения которых нельзя изменять, то их помечают модификатором final.

Если модификатором final в родительском классе был отмечен метод, то его переопределение в классах-наследниках запрещено. Именно так определены математические функции sin, cos и другие в классе Math. Используя метод Math.cos(x), можно быть уверенным, что вычисляется именно косинус числа x.

Если модификатором final пометить параметр метода, то его нельзя будет изменить внутри метода.

Если же пометить модификатором final весь класс, то его вообще нельзя будет наследовать. Именно так определен, например, класс Math:

```
public final class Math{ . . . }
```

Для переменных модификатор final имеет другой смысл. Поля, описанные с модификатором final, нельзя изменить. Таким образом в языке программирования Java определяются константы.

```
public final int MAX_VALUE = 10000;
```

По соглашению кодирования Java константы записываются прописными буквами, а слова в них разделяются знаком подчеркивания.

Константами можно пользоваться как переменными, доступными только для чтения. Попытка присвоить константе значение с помощью оператора присваивания вызывает ошибку компиляции.

Константы *e* и π в классе Math заданы следующим образом:

```
public static final double E = 2.7182818284590452354;
public static final double PI = 3.14159265358979323846;
```

В данном примере модификатор static означает, что это переменные класса, final – что изменять это значение нельзя (в том числе переопределять в классе-наследнике).

Ключевое слово super

При наследовании члены класса, имеющиеся в родительском классе, снова описывать не нужно – они наследуются.

Если в иерархии классов сигнатура метода класса-наследника совпадает с сигнатурой метода родительского класса, то метод класса-наследника переопределяет (замещает) метод родительского класса. Когда этот метод вызывается из своего класса, то выбирается метод, определённый в классе-наследнике, а метод из родительского класса будет скрыт.

Если нужно получить доступ к переопределённому методу из родительского класса, то используется ключевое слово super.

Ключевое слово super можно использовать для вызова конструктора родительского класса в конструкторе класса-наследника. В этом случае вызов конструктора родительского класса всегда должен быть первым оператором, выполняемым внутри конструктора класса-наследника. При вызове конструктора родительского класса с нужными аргументами, фактически, происходит инициализация переменных родительского класса, используя переданные значения соответствующих параметров. Остаётся инициализировать только новые добавленные поля класса-наследника.

У родительского класса может быть несколько перегруженных версий конструкторов, поэтому можно обращаться к конструкторам с разными параметрами. Выполняться будет тот конструктор, который соответствует указанным параметрам.

Поля не могут переопределяться. Если объявить в классе-наследнике поле с тем же именем, что и в родительском классе, то поле родительского класса скроется, к нему невозможно будет обратиться только по имени, нужно будет обязательно указать ключевое слово super.

Импорт класса

Если необходимо создать экземпляр класса или разработать класс-наследник пользовательского класса, который не содержится в данном пакете, то среда NetBeans не может импортировать пакет, так как он не входит в состав библиотек NetBeans, путь к которым находится автоматически.

Для указания пути в окне Проекты текущего пакета необходимо щелкнуть правой кнопкой мыши по папке Библиотеки (*Resources*) и в контекстном меню выбрать необходимое действие: Добавить проект, Добавить библиотеку или Добавить файл JAR (рис. 4.1).

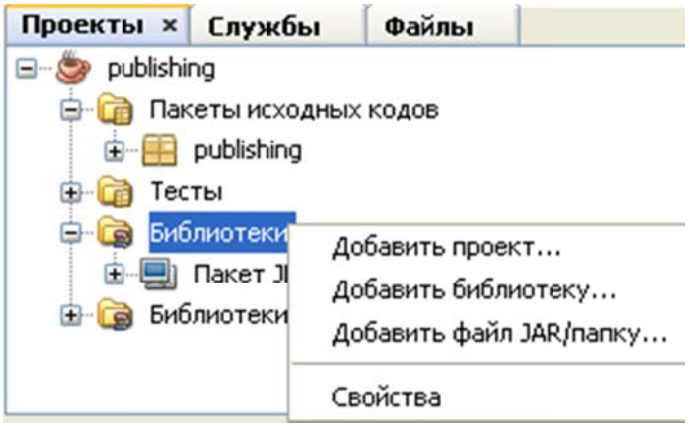

Рис. 4.1. Добавление классов в библиотеку

Далее необходимо указать месторасположение проекта или JAR-файла, а при добавлении библиотеки необходимо выбрать библиотеку из списка и нажать кнопку **Добавить библиотеку**.

Задание 8. Реализуйте наследование класса «Книга» (Book) от класса «Публикуемый материал» (Note). Опишите поля класса. Одним из полей должен быть порядковый номер. Создайте метод toString для вывода информации об экземпляре класса.

Класс Book будет иметь поля, которые должны быть у любой публикации: порядковый номер, название материала, фамилии и инициалы автора, объем материала (в авторских листах). Кроме этого, в выходных сведениях книги указывается название издательства, город и год выпуска, количество страниц.

Имена и типы полей класса Book представлены в табл. 4.1.

Таблица 4.1

Имена и типы полей класса Book

Имя поля	Тип поля	Описание
numberBook	int	Порядковый номер материала
bookTitle	String	Название материала
authorBook	String	Автор материала
numberAuthorsSheets	double	Количество авторских листов
publishingHouse	String	Название издательства
publishingLocation	String	Город
publishingYear	int	Год издания
numberPage	int	Кол-во страниц

Как видно из табл. 4.1, первые четыре поля у класса Book аналогичны полям класса Note, поэтому логично не создавать эти поля заново, а сделать класс Book наследником класса Note. Для того чтобы класс Book мог использовать

поля numberNote, noteTitle, authorNote и numberAuthorsSheets класса Note, необходимо в классе Note изменить модификатор доступа этих полей с private на protected.

Порядок работы

1. Перейдите на вкладку класса Note.
2. Для полей numberNote, noteTitle, authorNote и numberAuthorsSheets измените модификатор доступа с private на protected.
3. Сохраните изменения.
4. В проекте publishing создайте новый класс Book.
5. Удалите комментарии, расположенные выше заголовка проекта.
6. В комментариях к классу запишите автора и назначение класса.
7. После заголовка класса объявите поля класса (только новые поля; доступ к старым полям можно будет получить из родительского класса):

```
private String publishingHouse; // Издательство
private String publishingLocation; // Город
private int publishingYear; // Год издания
private int numberPage; // Кол-во страниц
```

8. Создайте конструктор класса Book. В качестве формальных параметров укажите также и поля родительского класса. Первой командой конструктора будет вызов конструктора родительского класса:

```
/**
 * Конструктор для создания экземпляра книги
 * @param noteTitle - название
 * @param authorNote - автор
 * @param numberAuthorsSheets - количество авторских листов
 * @param publishingHouse - издательство
 * @param publishingLocation - город
 * @param publishingYear - год издания
 */
public Book(String noteTitle, String authorNote,
     double numberAuthorsSheets, String publishingHouse,
     String publishingLocation, int publishingYear) {
 super(noteTitle, authorNote, numberAuthorsSheets);
 this.publishingHouse = publishingHouse;
 this.publishingLocation = publishingLocation;
 this.publishingYear = publishingYear;
}
```

12. Создайте метод toString:

```
/**
 * Метод, формирующий информацию о книге
 * @return the str - возвращает строку
 */
```

```
@Override
public String toString() {
   String str = numberNote + " " + authorNote + " " + noteTitle
        + ". - " + publishingLocation + ".: " + publishingHouse
        + ", " + publishingYear + ".- " + getNumberPages()
        + " с.";
   return str;
}
```

13. Создайте методы get*** для всех полей класса Book.

14. Для поля publishingYear создайте метод setPublishingYear.

15. Сохраните класс Book.

16. Перейдите в класс Publishing. Закомментируйте строки изменения телефона автора, вывода его фамилии с инициалами, а также проверки методов set*** класса Note.

17. Так как издательство одно и находится в Минске, то в классе Publishing создадим две константы для указания названия издательства и города:

```
public class Publishing {
      private static final String PUBLISHING_HOUSE = "Эрудит";
      private static final String PUBLISHING_LOCATION = "Мн";
      . . .
```

18. Для проверки правильности работы класса Book создайте экземпляр этого класса и выведите на консоль информацию о нем. Для этого в метод main добавьте следующие команды:

```
// Создание книг
System.out.println("\nКниги:");
Book bookVasilev3 = new Book("Программирование в Java",
        shortNameVasilev, 30.5, PUBLISHING_HOUSE,
        PUBLISHING_LOCATION, 2016);
System.out.println(bookVasilev3.toString());
```

19. Выполните проект. На консоль будет выведена информация:

```
run:
Авторы:
Васильев Павел Иванович, паспортные данные: MP2035648, выдан 18.05.2011,
    адрес: г.Минск, ул.Васнецова, д.45, кв.79, конт. телефон: 8(029)33-564-78-02

Публикуемые материалы:
1 Васильев П.И. Создание классов в Java, авт. листов: 3.6
Кол-во страниц 39
2 Васильев П.И. Среда NetBeans, авт. листов: 2.8
Кол-во страниц 30
```

```
Книги:
3 Васильев П.И. Программирование в Java. - Мн.: Эрудит, 2016.-
366 с.
СБОРКА УСПЕШНО ЗАВЕРШЕНА (общее время: 0 секунд)
```

9. Обратите внимание, что номер книги сформировался верно: в конструкторе класса Book был вызван конструктор родительского класса Note.

10. В качестве дополнительного задания можно создать еще одного автора, его книгу и вывести эту информацию на консоль.

Краткие итоги

Родительский класс реализуется для группы объектов, имеющих одинаковые свойства и общее поведение. Различия описываются в дочерних классах.

Наследование позволяет разрабатывать новые классы на основе родительского , добавляя в них поля и методы. Базовый класс называется родителем (или суперклассом), новые классы – его потомками (наследниками или подклассами). От потомков можно наследовать далее, получая очередных потомков.

Набор классов, связанных отношением наследования, называется иерархией классов. Класс, стоящий во главе иерархии, называется базовым классом иерархии.

Структура класса-наследника имеет следующий формат:

```
Модификаторы class Имя_Дочернего_Класса extends Имя_Класса_Ро-
дителя {
        объявление полей;
        объявление конструкторов;
        объявление методов;
}
```

С помощью аннотации @Override в классе-наследнике можно переопределять методы родительского класса.

Для запрета наследования члены класса помечаются модификатором final.

Если члены класса-наследника совпадают с членами родительского класса, то происходит переопределение (замещение) членов родительского класса. Для обращения к членам родительского класса, скрытым членами класса-наследника, используется ключевое слово super.

5. АГРЕГАЦИЯ И КОМПОЗИЦИЯ. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Механизм наследования позволяет использовать уже существующий код, но он используется для классов, сходных по функционалу. В языке Java есть и другие возможности разработки нового класса с использованием уже существующих классов: агрегация и композиция.

Под агрегацией понимают методику создания нового класса из объектов уже существующих классов.

Агрегация (*aggregation*) – отношение «часть-целое» между двумя равноправными объектами, когда один объект имеет ссылку на другой объект. Оба объекта могут существовать независимо: если один из них будет уничтожен, то второй – нет.

Реализация агрегации производится путем ссылки на экземпляр другого класса. Например, если в автомобиле имеется четыре колеса, тогда в классе Car будет массив из четырех ссылок на экземпляры класса Wheel.

Композиция (*composition*) – более строгий вариант агрегации, когда включаемый объект может существовать только как часть класса. Если класс будет уничтожен, то и включённый объект тоже будет уничтожен.

Например, в библиотеке имеются читатели, тогда в классе Library будет массив (или список) экземпляров класса `ReadersCard` (карточка читателя). Карточки читателей имеет смысл использовать только в конкретной библиотеке.

Задание 9. Реализуйте принцип агрегации путем создания класса «Журнал» (Magazine), который должен содержать несколько статей (до 10). Опишите поля класса.

Поля класса Magazine описаны в табл. 5.1.

Таблица 5.1

Имена и типы полей класса Magazine

Имя поля	Тип поля	Описание
magazineTitle	String	Название журнала
numberMagazine	int	Номер выпуска (от 1 до 12)
publishingYear	int	Год издания
notes	Note	Массив статей
publishingHouse	String	Название издательства
publishingLocation	String	Город
numberPage	int	Количество страниц

Порядок работы

1. В проекте publishing создайте новый класс Magazine.
2. Удалите комментарии, расположенные выше заголовка проекта.
3. В комментариях к классу запишите автора и назначение класса.
4. После заголовка класса объявите поля класса:

```
private String magazineTitle;        // название журнала
private int numberMagazine;          // номер выпуска
private int publishingYear;          // Год издания
private Note notes [];               // массив статей
private String publishingHouse;      // Издательство
private String publishingLocation;   // Город
private int numberPage;              // Кол-во страниц
```

5. Создайте конструктор класса Magazine.

```
/**
 * Конструктор для создания экземпляра журнала
 * @param magazineTitle - название
 * @param numberMagazine - номер выпуска
 * @param publishingYear - год издания
 * @param notes - массив статей
 * @param publishingHouse - издательство
 * @param publishingLocation - город
 */
public Magazine(String magazineTitle, int numberMagazine,
      int publishingYear, Note[] notes, String publishingHouse,
      String publishingLocation) {
    this.magazineTitle = magazineTitle;
    this.numberMagazine = numberMagazine;
    this.publishingYear = publishingYear;
    this.notes = notes;
    this.publishingHouse = publishingHouse;
    this.publishingLocation = publishingLocation;
}
```

20. Создайте методы get*** для всех полей класса Magazine.

21. Для того чтобы можно было изменять содержание журнала, создайте метод setNotes.

22. Для проверки правильности класса Magazine создайте метод toString:

```
/**
 * Метод, формирующий информацию о журнале
 * @return str - возвращает строку
 */
@Override
public String toString() {
   String str = "Журнал " + magazineTitle + " № " +
```

```
            numberMagazine + " " + publishingYear + ". - "
            + publishingLocation + ".: " + publishingHouse;
   for (int i=0; i<notes.length; i++)
      if (notes[i] != null)
           str += "\n" + notes[i].toString();
   return str;
}
```

23. Сохраните класс Magazine.

24. Перейдите в класс Publishing.

25. Для проверки правильности работы класса Magazine создайте экземпляр этого класса, массив статей, публикуемых в журнале, и выведите на консоль информацию о журнале. Для этого в метод main добавьте следующие команды:

```
// Создание журнала
System.out.println("\nЖурналы:");
Note notes [] = new Note [10]; // массив статей
notes[0] = noteVasilev1;
notes[1] = noteVasilev2;
Magazine magazine = new Magazine("Программирование", 1, 2016,
       notes, PUBLISHING_HOUSE, PUBLISHING_LOCATION);
System.out.println(magazine.toString());
```

26. Выполните проект. На консоль будет выведена информация:

```
. . .
Журналы:
Журнал Программирование № 1 2016. - Мн.: Эрудит
1 Васильев П.И. Создание классов в Java, авт. листов: 3.6
2 Васильев П.И. Среда NetBeans, авт. листов: 2.8
СБОРКА УСПЕШНО ЗАВЕРШЕНА (общее время: 0 секунд)
```

В качестве дополнительного задания можно создать еще одного автора, еще несколько статей, сформировать второй выпуск журнала и вывести эту информацию на консоль.

Задание 10. Реализуйте принцип композиции путем создания класса «Содержание» (Contents), который формирует оглавление журнала. Опишите поля класса. Создайте метод для создания оглавления, метод для подсчета общего количества страниц в журнале и метод toString.

Поля класса Contents описаны в табл. 5.2.

Таблица 5.2

Имена и типы полей класса Contents

Имя поля	Тип поля	Описание
notes	Note	массив статей
pageNumberNote	int	двумерный массив номеров страниц (оглавление)
numberPage	int	текущий номер страницы

Порядок работы

1. В проекте publishing создайте новый класс Contents.
2. Удалите комментарии, расположенные выше заголовка проекта.
3. В комментариях к классу запишите автора и назначение класса.
4. После заголовка класса объявите его поля:

```
private Note notes [];                // массив статей
private int pageNumberNote [] [];     // массив номеров страниц
private int numberPage;               // текущий номер страницы
```

5. Создайте конструктор класса Contents. Массив pageNumberNote для каждой статьи содержит две характеристики: номер начальной страницы и количество занимаемых страниц. Первая страница журнала – титульная, вторая – содержит оглавление. Таким образом, публикации начинаются с третьей страницы.

```
/**
 * Конструктор для создания содержания журнала
 * @param notes - массив статей
 */
public Contents(Note[] notes) {
     this.notes = notes;
     // Массив pageNumberNote содержит для каждого материала:
     // 1. Номер начальной страницы материала
     // 2. Кол-во страниц, необходимых для размещения материала
     pageNumberNote = new int [notes.length] [2];
     createContents(notes);
}
```

6. Создайте метод createContents для формирования оглавления журнала:

```
/**
 * Метод формирования оглавления
 */
public void createContents(Note[] notes){
     int i;
     // Очистка оглавления
     for (i = 0; i < notes.length; i++){
         pageNumberNote[i][0] = 0;
         pageNumberNote[i][1] = 0;
     }
     // первая страница - титульная
     // вторая страница содержит оглавление
     // материалы начинаются с третьей страницы
     numberPage = 3;
     i = 0;
     while (notes[i] != null){
          pageNumberNote[i][0] = numberPage; // начальный номер
          // объем материала
```

```
            pageNumberNote[i][1] = notes [i].getNumberPages();
            // корректировка текущего номера страниц
            numberPage += pageNumberNote[i][1];
            i++;
        }
    }
```

7. Создайте метод getNumberPages для подсчета общего количества страниц журнала. Переменная numberPage содержит номер очередной пустой страницы, поэтому количество страниц в журнале будет на единицу меньше:

```
/**
 * Метод подсчета общего количества страниц в журнале
 * @return int - Кол-во страниц
 */
public int getNumberPages() {
    return numberPage - 1;
}
```

8. Создайте метод toString:

```
/**
 * Метод, выводящий оглавление
 * @return the str - возвращает строку
 */
@Override
public String toString() {
   String str = "\nСодержание:";
   int i = 0;
   while (notes[i] != null){
      str += "\n" + (i+1) + " " + notes[i].getAuthorNote() +
   " " + notes[i].getNoteTitle() + "\t" + pageNumberNote[i][0];
      i++;
   }
   return str;
}
```

9. Сохраните класс Contents.

10. Таким образом, в конструкторе класса Contents создается массив, необходимый для вывода оглавления журнала, а само оглавление формируется в методе createContents.

11. Класс Contents необходимо связать с классом Magazine. Для этого перейдите в класс Magazine. В конструктор класса последней строкой добавьте команду создания экземпляра класса Contents:

```
contents = new Contents(notes);
```

12. Для вывода оглавления на консоль необходимо изменить метод toString класса Magazine:

```
    public String toString() {
         String str = "Журнал " + magazineTitle + " № " + number-
Magazine +
             " " + publishingYear + ". - " + publishingLocation +
".: "
             + publishingHouse;
         str += contents.toString();
         return str;
    }
```

27. Если редактор захочет изменить содержимое журнала (добавить или удалить какой-либо материал), то оглавление необходимо изменить. Поэтому измените метод setNotes класса Magazine:

```
    /**
     * Метод для изменения массива статей журнала
     * @param notes the notes to set
     */
    public void setNotes(Note[] notes) {
        this.notes = notes;
        contents.createContents(notes);
    }
```

28. Для проверки правильности вывода оглавления журнала выполните проект. На консоль будет выведена информация:

```
    . . .
    Журнал Программирование № 1 2016. - Мн.: Эрудит
    Содержание:
    1 Васильев П.И. Создание классов в Java     3
    2 Васильев П.И. Среда NetBeans              42
```

13. Можно узнать общее количество страниц в журнале. Для этого в метод main класса Publishing добавьте команду:

```
    System.out.println("Всего страниц: " + magazine.getNumberPage());
```

14. Выполните проект. На консоль будет выведена информация:

```
    . . .
    Журнал Программирование № 1 2016. - Мн.: Эрудит
    Содержание:
    1 Васильев П.И. Создание классов в Java    3
    2 Васильев П.И. Среда NetBeans             42
    Всего страниц: 71
    СБОРКА УСПЕШНО ЗАВЕРШЕНА (общее время: 0 секунд)
```

Краткие итоги

Агрегация – это такое отношение между классами, когда в одном классе имеется ссылка на экземпляр другого класса. В то же время экземпляры обоих классов могут существовать независимо друг от друга.

Композиция – это такое отношение между классами, когда в одном классе создается экземпляр другого класса. Если экземпляр первого класса будет уничтожен, то и включённый в него экземпляр другого класса тоже будет уничтожен.

В рассмотренном примере журнал составлялся из статей, поступающих в издательство. Независимо от того, будут ли опубликованы статьи в журнале, они существуют как самостоятельные объекты. В данном случае использовалась методика агрегации.

Оглавление создается в каждом номере журнале. Оно не существует самостоятельно, его использование вне журнала бессмысленно, поэтому здесь использовался принцип композиции.

6. ОТНОШЕНИЯ МЕЖДУ КЛАССАМИ НА UML-ДИАГРАММАХ

При проектировании системы необходимо не только идентифицировать сущности в виде классов, но и указать, как они соотносятся друг с другом. На языке UML способы, которыми элементы связаны друг с другом, моделируются в виде отношений.

Отношение (*Relationship*) – связь между элементами. В объектно-ориентированном проектировании особое значение имеют четыре типа отношений: зависимости, обобщения, реализации и ассоциации. Графически отношение представлено линией, тип которой зависит от вида отношения (рис. 6.1).

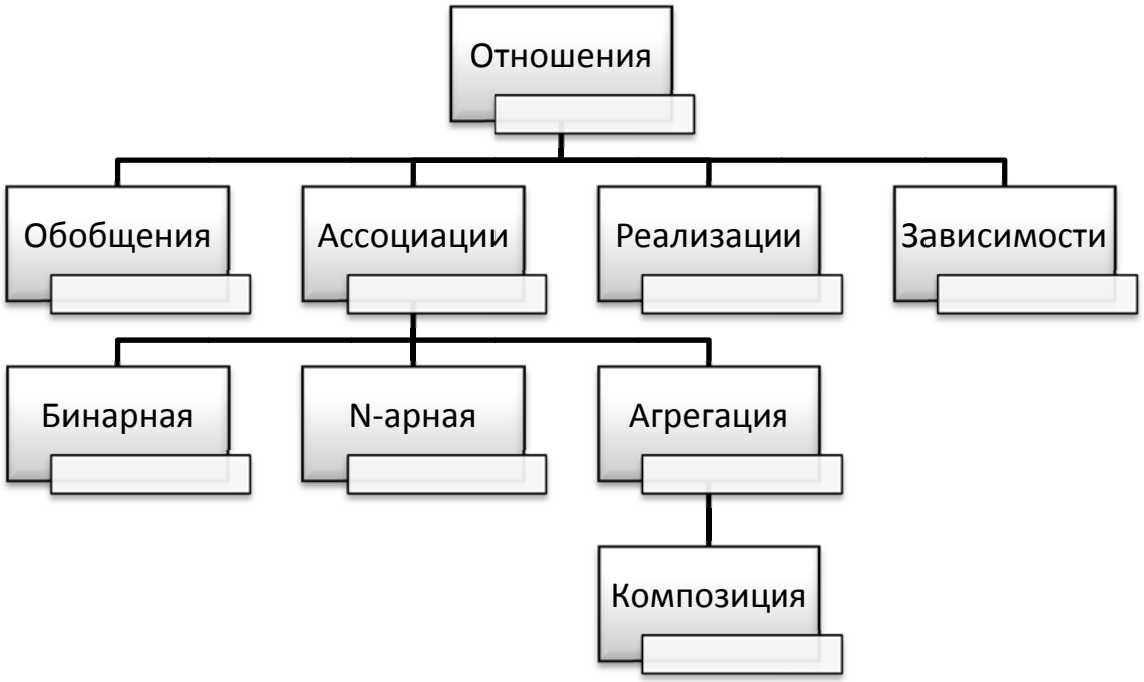

Рис. 6.1. Классификация отношений

Зависимостью (*Dependency*) называется отношение использования, определяющее, что изменение состояния объекта одного класса (независимого) может повлиять на объект другого класса (зависимого), который его использует, причем обратное в общем случае неверно. Зависимости применяются тогда, когда экземпляр одного класса использует экземпляр другого, например, в качестве параметра метода.

Графически зависимость изображается пунктирной линией со стрелкой, направленной от элемента к тому, от которого он зависит (рис. 6.2).

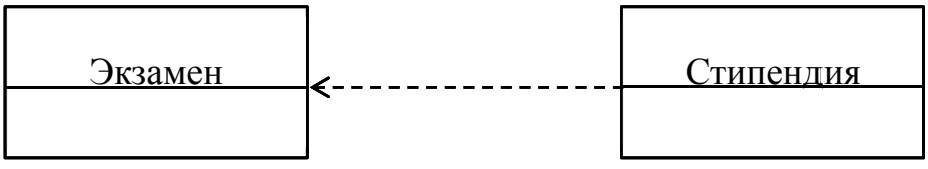

Рис. 6.2. Отношение зависимости

Обобщение (*Generalization*) – это отношение между общей сущностью (суперклассом или родителем) и ее конкретным воплощением (подклассом или потомком).

Графически отношение обобщения изображается в виде линии с незакрашенной стрелкой, указывающей на родителя (рис. 6.3). Отношение обобщения реализуется при наследовании классов.

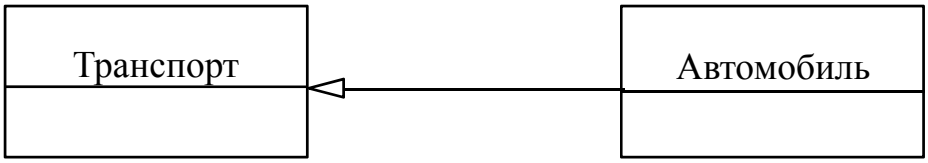

Рис. 6.3. Отношение обобщения

Реализацией (*Realization*) называется отношение между классификаторами (классами, интерфейсами), при котором один описывает контракт (интерфейс сущности), а другой гарантирует его выполнение. Это отношение между спецификацией и ее программной реализацией; указание на то, что поведение наследуется без структуры.

Ассоциация (*Association*) показывает, что объект одного класса связан с объектом другого класса и отражает некоторое отношение между ними. В этом случае можно перемещаться (с помощью вызова методов) от объекта одного класса к объекту другого.

Ассоциация – структурное отношение, описывающее множество связей между объектами. Ассоциации являются как бы клеем, который связывает систему воедино. Без ассоциаций имелось бы просто некоторое количество классов, не способных взаимодействовать друг с другом.

Ассоциация, связывающая два класса, называется бинарной. Иногда бывает необходимо, создавать ассоциации, связывающие сразу несколько классов. В этом случае они называются n-арными.

Графически ассоциация изображается в виде сплошной прямой линии со стрелкой, направленной от исходного класса к целевому (рис. 6.4).

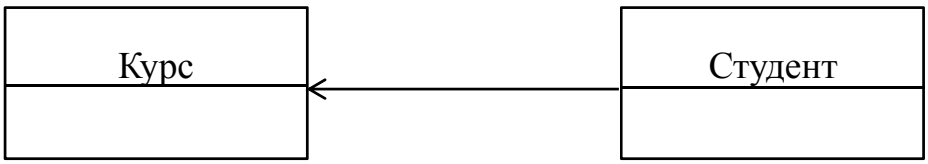

Рис. 6.4. Отношение ассоциации

Одним из вариантов отношения ассоциации является агрегация.

Агрегация – ассоциация, моделирующая взаимосвязь «часть/целое» между классами, которые в то же время могут быть равноправными. Оба класса при этом находятся на одном концептуальном уровне, и ни один не является более важным, чем другой.

Агрегация изображается в виде сплошной линии с незакрашенным ромбом со стороны «целого», как показано на рис. 6.5.

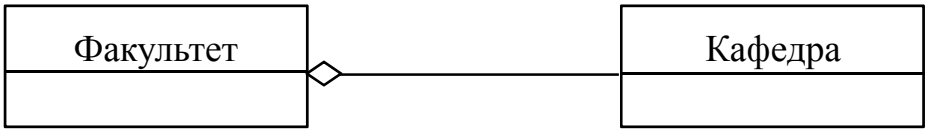

Рис. 6.5. Отношение агрегации

Композиция – более строгий вариант агрегации. Композиция имеет жёсткую зависимость времени существования экземпляров класса-контейнера и экземпляров содержащихся в нем классов. Если контейнер будет уничтожен, то всё его содержимое будет также уничтожено. Графически представляется, как и агрегация, но с закрашенным ромбом (рис. 6.6).

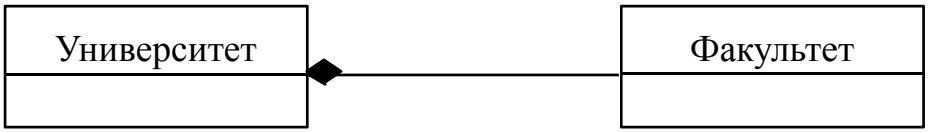

Рис. 6.6. Отношение композиции

Существуют два подхода к проектированию UML-диаграмм:

1. С помощью средств прямого проектирования (*Forward Engineering*) можно создавать и изменять исходный код классов путем создания и редактирования UML-диаграмм.

2. С помощью средств обратного проектирования (*Reverse Engineering*) можно создавать UML-диаграммы классов, анализируя программный код классов.

Краткие итоги

Диаграмма классов отображает множество классов и отношения между ними.

Классы могут находиться в различных отношениях: зависимости, обобщения, реализации и ассоциации. Графически это изображается на диаграмме с помощью линий разного типа (сплошной, пунктирной, со стрелкой и т. д.).

UML-диаграмма классов, отображающая отношения между классами в проекте Publishing, изображена на рис. 6.7.

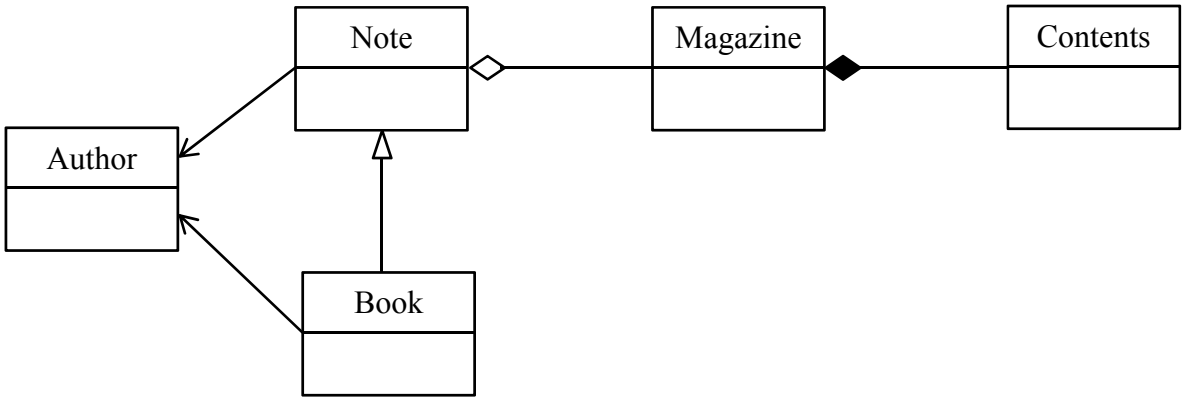

Рис. 6.7. Отношения между классами в проекте Publishing

7. ПОЛИМОРФИЗМ. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Перегрузка методов

В языке программирования Java каждый метод обладает определенной сигнатурой, которая представляет собой совокупность имени и формальных параметров. Два метода могут иметь одинаковые имена, если их сигнатуры отличаются по количеству или типам формальных параметров. Это называют перегрузкой (*overloading*). Компилятор по количеству или типу фактических параметров ищет тот из существующих методов, сигнатура которого соответствует сигнатуре метода при обращении к нему.

Перегрузка методов (*overloading*) – это создание нескольких методов с одинаковыми именами, но с различными сигнатурами, по которым эти методы отличаются друг от друга.

Методы называют перегруженными, если два или более методов класса имеют одно имя, но их формальные параметры не совпадают по количеству или типам.

Явление, когда разный программный код связан с одним и тем же именем, называется проявлением полиморфизма (одно имя, но много форм). Полиморфизм позволяет упростить использование создаваемых классов.

Перегрузка конструкторов

В классах часто объявляется несколько конструкторов. Такие конструкторы также называются перегруженными. Они отличаются списком параметров. Перегрузка конструкторов используется, если при создании экземпляров класса необходимо указать различное количество полей.

Задание 11. Издательство может сотрудничать с иностранными авторами, у которых нет отчества. Поэтому для класса Author разработайте перегруженный метод getShortName, который формирует короткую строку, состоящую из фамилии и первой буквы имени автора.

Порядок работы

1. Перейдите на вкладку класса Author.
2. После метода getShortName запишите код перегруженного метода:

```
/**
 * Перегруженный метод, возвращающий фамилию и инициалы автора
 * @param surname - фамилия
 * @param name - имя
 * @return the str - возвращает строку типа Иванов И.
 */
String getShortName(String surname, String name) {
```

```
        String str = surname + " " + name.substring(0, 1) + ".";
        return str;
    }
```

3. Перейдите на вкладку класса Publishing.

4. Для проверки правильности работы перегруженного метода создайте автора без отчества и выведите на консоль информацию о нем. Для этого в методе main добавьте следующие команды:

```
    Author vilson = new Author("Вилсон", "Альгис", "",
            "MP4503878, выдан 02.11.2001",
            "г. Минск, ул. Плеханова, д. 112, кв. 125",
            "8(029)44-854-05-91");
    System.out.println(vilson.toString());
    String shortNameVilson = vil-
son.getShortName(vilson.getSurname(),
            vilson.getName());
    System.out.println("Фамилия и инициалы: " + shortNameVilson);
```

5. Выполните проект. На консоль будет выведена информация:

```
    . . .
    Вилсон Альгис, паспортные данные: MP4503878, выдан 02.11.2001,
    адрес: г.Минск, ул.Плеханова, д.112, кв.125, конт. телефон:
8(029)44-854-05-91
    Фамилия и инициалы: Вилсон А.
```

Задание 12. У материалов, поступающих в издательство, может быть несколько авторов. Для класса Note разработайте два перегруженных конструктора: для двух и трех авторов.

Порядок работы

1. Перейдите на вкладку класса Note.

2. После кода первого конструктора (для одного автора) запишите код конструктора для материала двух авторов. Сигнатура конструктора будет содержать на один формальный параметр больше. В теле конструктора при формировании поля authorNote класса Note склейте две строки с описанием авторов:

```
    /**
     * Конструктор для создания экземпляра публикуемого материала
     * двух авторов
     * @param noteTitle - название материала
     * @param author1Note - 1 автор материала
     * @param author2Note - 2 автор материала
     * @param numberAuthorsSheets - количество авторских листов
     */
    public Note(String noteTitle, String author1Note, String
            author2Note, double numberAuthorsSheets) {
       numberNote = ++lastNumberNote; // порядковый номер материала
```

```
        this.noteTitle = noteTitle;
        this.authorNote = author1Note + ", " + author2Note;
        this.numberAuthorsSheets = numberAuthorsSheets;
    }
```

3. По аналогии напишите программный код конструктора для материала трех авторов.

4. Для проверки правильности работы перегруженного конструктора класса Note создайте экземпляр статьи двух авторов и выведите на консоль информацию о нем. Для этого в метод main класса Publishing добавьте следующие команды:

```
// Перегрузка конструкторов
System.out.println("\nСтатья двух авторов:");
Note noteVasilev3 = new Note("Реализация принципов ООП",
        shortNameVasilev, shortNamePetrov, 2.4);
System.out.println(noteVasilev3.toString());
```

5. Выполните проект. На консоль будет выведена информация:

```
. . .
Статья двух авторов:
5 Васильев П.И., Петров В.С. Реализация принципов ООП, авт. ли-
стов: 2.4
```

Задание 13. У книг, поступающих в издательство, также может быть несколько авторов. Для класса Book разработайте два перегруженных конструктора: для двух и трех авторов.

Порядок работы

1. Перейдите на вкладку класса Book.

2. После кода первого конструктора класса запишите код конструктора для книги двух авторов. Сигнатура конструктора будет содержать на один формальный параметр больше. В теле конструктора используется перегруженный конструктор класса Note для материала двух авторов:

```
/**
 * Конструктор для создания экземпляра книги
 * двух авторов
 * @param noteTitle - название
 * @param author1Note - автор 1
 * @param author2Note - автор 2
 * @param numberAuthorsSheets - количество авторских листов
 * @param publishingHouse - издательство
 * @param publishingLocation - город
 * @param publishingYear - год издания
 */
public Book(String noteTitle, String author1Note,
        String author2Note, double numberAuthorsSheets,
        String publishingHouse, String publishingLocation,
```

```
            int publishingYear) {
        super(noteTitle, author1Note, author2Note, numberAuthorsSheets);
        this.publishingHouse = publishingHouse;
        this.publishingLocation = publishingLocation;
        this.publishingYear = publishingYear;
        numberPage = getNumberPages();
    }
```

3. По аналогии напишите программный код конструктора для книги трех авторов.

4. Перейдите на вкладку класса Publishing.

5. Для проверки правильности работы перегруженного конструктора класса Book создайте новую книгу трех авторов и выведите на консоль информацию о ней. Для этого в метод main добавьте следующие команды:

```
System.out.println("\nКнига трех авторов:");
Book book5 = new Book("Современные языки программирования",
     shortNameVasilev, shortNameVilson, shortNamePetrov,
       22.5, PUBLISHING_HOUSE, PUBLISHING_LOCATION, 2016);
System.out.println(book5.toString());
```

6. Выполните проект. На консоль будет выведена информация:

```
. . .
Книга трех авторов:
6 Васильев П.И., Вилсон А., Петров В.С. Современные языки программирования. - Мн.: Эрудит, 2016.- 243 с.
```

Краткие итоги

В классе можно создавать конструкторы и методы с одинаковыми именами, но с различными сигнатурами. Такие методы и конструкторы называются перегруженными.

ПРИЛОЖЕНИЕ 1

Среда NetBeans 7.4

NetBeans – это интегрированная среда разработки (*Integrated Development Environment*, IDE), с помощью которой можно набирать и редактировать тексты программ, компилировать, подключать нужные библиотеки, запускать и отлаживать программный код.

После запуска среды на экране появится окно, изображенное на рис. П1.1.

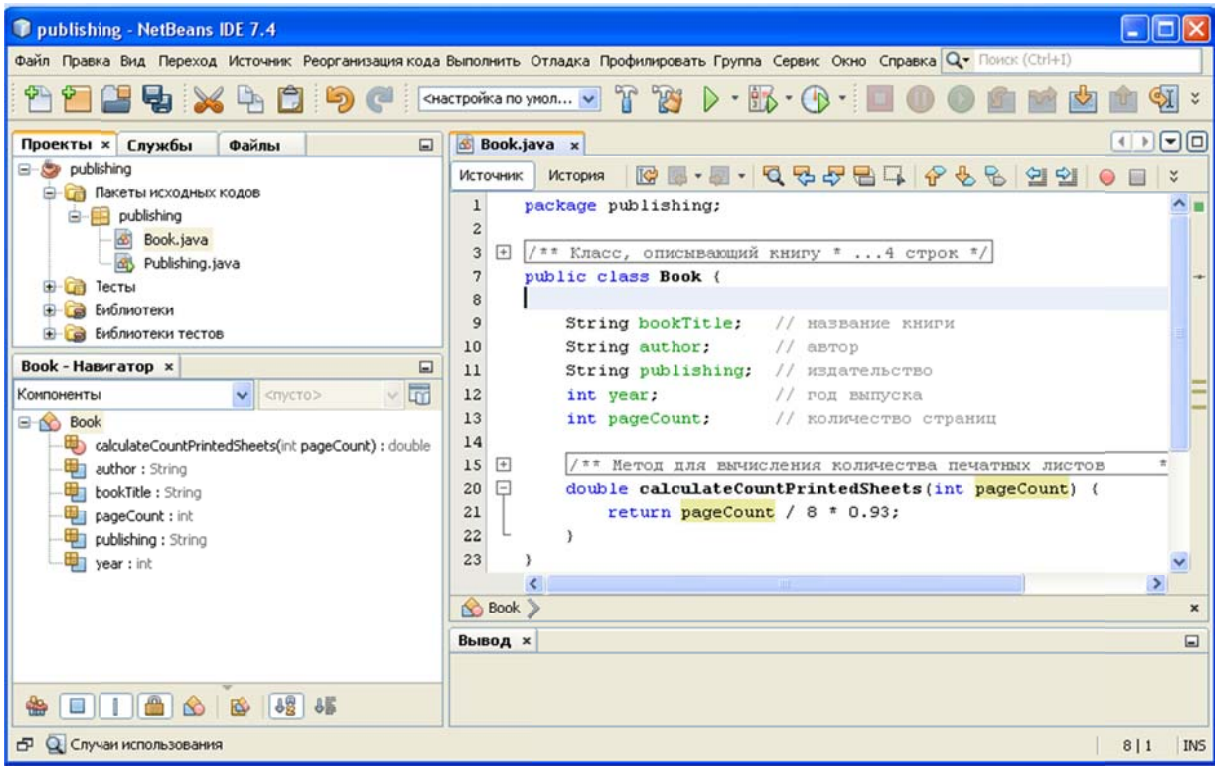

Рис. П1.1. Общий вид окна NetBeans IDE 7.4

В левом верхнем окне Проекты (*Projects*) отображается дерево проектов. На рисунке – это дерево для проекта publishing. В этом окне может отображаться одновременно несколько проектов. Узлы деревьев сворачиваются нажатием на значок с минусом и разворачиваются – на значок с плюсом.

Для редактирования исходного кода какого-либо класса в NetBeans необходимо открыть проект содержащий этот класс, затем в окне Проекты нажатием на плюсики развернуть последовательность узлов данного проекта (см. рис. П1.1). Затем нужно сделать двойной щелчок левой кнопкой мыши по имени java-файла или выбрать команду Открыть (*Open*) с помощью контекстного меню.

Если вы открываете новый проект, старый проект не закрывается и в дереве проектов видны все доступные проекты. Это позволяет работать сразу с несколькими проектами одновременно.

В левом нижнем окне Навигатор (*Navigator*) выводится список имен членов класса, редактируемого в данный момент: имена полей, методов и кон-

структоров. В этом окне двойной щелчок по имени поля или метода приводит к тому, что в окне редактора исходного кода происходит переход на то место, где расположено соответствующее поле или метод. Кнопки в нижней части окна предназначены для управления содержимым:

– показывают унаследованные члены;

– показывают поля;

– показывают статические элементы;

– показывают внутренние элементы;

– показывают внутренние классы;

– отображают полные имена;

– сортируют по имени;

– сортируют по источнику.

В главном окне на вкладке Источник (*Source*) находится редактор исходного кода проекта, в котором отображается программный код проекта. Правее от вкладок находится панель инструментов редактора кода, изображенная на рис. П1.2.

Рис. П1.2. Панель инструментов редактора кода

Группа кнопок  используется для поиска искомого элемента по тексту программы.

Группа кнопок  используется для установки закладок и быстрого перемещения по закладкам вверх и вниз

Кнопки  используются для смещения строк кода влево/вправо.

Кнопки  используются для «закомментирования» / отмены выделенных строк программного кода.

В редакторе исходного кода синтаксические конструкции подсвечиваются разными цветами и выделяются разными стилями. Например, поля отображаются зеленым шрифтом, ключевые слова – синим, имена классов – жирным шрифтом черного цвета. Синтаксические ошибки подчеркиваются красной волнистой линией; при наведении мыши появляется всплывающая подсказка.

В правом нижнем окне Вывод (*Output*) выводится служебная информация о ходе компиляции и запуске приложения, а также выводится информация из приложения, если в нем был предусмотрен консольный вывод.

Порядок создания пакета

Все классы Java распределяются по пакетам (*package*). Каждый пакет может содержать один и более классов.

Порядок создания нового пакета в среде NetBeans может быть следующим:

1. Запустите NetBeans.

2. Нажмите кнопку (оранжевого цвета) на панели инструментов или откройте меню Файл → Создать проект (*File* → *New Project*).

3. Выберите Категорию: Java (*Categories*: *Java*); Проекты: Приложение Java (*Projects*: *Java Application*) и нажмите кнопку **Далее >** (*Next >*).

4. Введите имя проекта, место его расположения и нажмите кнопку **Готово** (*Finish*).

При этом по умолчанию в пакете будет создан одноименный класс.

Порядок создания класса

1. Для добавления нового класса в открытый проект к уже существующим классам, необходимо:

1.1. Нажать кнопку (белого цвета) на панели инструментов или открыть меню Файл → Создать файл (*File* → *New file*).

1.2. Выбрать Категорию: Java (*Categories*: *Java*); Тип файла: Класс Java (*Java class*) и нажать кнопку **Далее >** (*Next >*).

1.3. Ввести имя класса и нажать кнопку **Готово** (*Finish*).

2. В комментариях к классу записать назначение класса, автора проекта.

3. Объявить поля класса.

4. Создать необходимые методы класса.

5. Разработать необходимые конструкторы для создания экземпляров класса.

Форматирование программного кода

Для лучшей читаемости программного кода его необходимо правильно форматировать. Строки, содержащие команды, находящиеся в блоке тела цикла, метода или класса, должны быть сдвинуты вправо на 4 позиции. Это можно сделать с помощью кнопок или команды Формат меню Источник (*Source* → *Format*) (рис. П1.3).

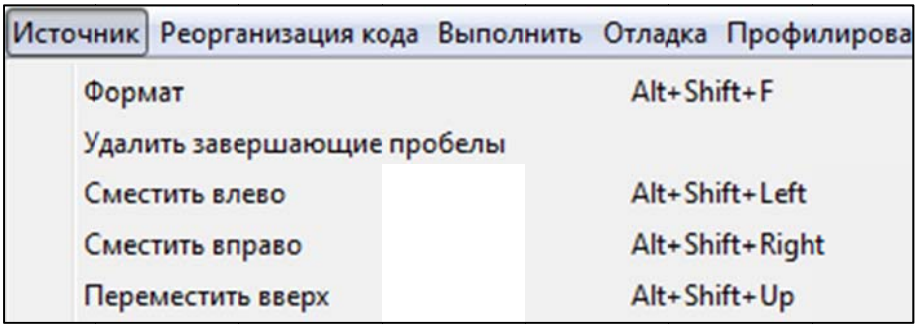

Рис. П1.3. Меню «Источник»

Компиляция файлов проекта и запуск приложения

В среде NetBeans предусмотрены два режима компиляции: скомпилировать (*compile*) и построить (*build*).

Для компиляции проекта следует открыть меню Выполнить (рис. П1.4) и выбрать команду Собрать проект (*Run* → *Build project*). Можно нажать клавишу **F11** или кнопку  на панели инструментов. При этом из исходных кодов будут заново скомпилированы все классы проекта.

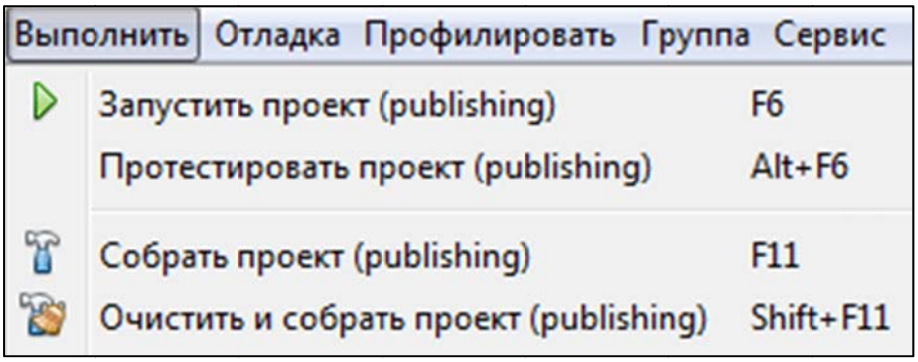

Рис. П1.4. Меню «Выполнить»

По команде Очистить и собрать проект из меню Выполнить (*Run* → *Clean and build project*) удаляются все выходные файлы проекта (очищаются папки build и dist), после чего заново компилируются все классы проекта. Эту операцию можно выполнить, нажав комбинацию клавиш **Shift** + **F11** или кнопку  на панели инструментов.

Для того чтобы запустить приложение из среды NetBeans, следует выбрать команду Запустить проект (*Run* → *Run project*) в меню Выполнить. Можно нажать клавишу **F6** или кнопку с зеленым треугольником  на панели инструментов. При запуске приложение всегда автоматически компилируется, так что после внесения изменений для запуска обычно достаточно нажать клавишу **F6**.

После запуска приложения на выходной консоли появится служебная информация о ходе компиляции и запуске.

Генерация документации с помощью утилиты javadoc

Одним из важнейших требований к написанию программного кода является документирование создаваемого кода. В языке программирования Java для этих целей применяются специальные комментарии – комментарии документации (javadoc).

Комментарии документации начинаются с комбинации символов `/**` и заканчиваются комбинацией символов `*/`. Каждая внутренняя строка начинается с символа * и может содержать специальные теги, которые начинаются с символа @.

Комментарии документации создаются для классов, интерфейсов, конструкторов, методов, полей и размещаются непосредственно перед их описанием. Такие комментарии приведены в примерах пособия.

Средством обработки комментариев, внедренных в исходный код, и создания для класса справочных HTML-файлов является инструмент javadoc, входящий в состав JDK. В среде NetBeans для генерации документации используют команду главного меню: Выполнить → Создать документацию Java (…) (*Run → Generate Javadoc* (...)) (вместо многоточия будет стоять имя открытого проекта). При этом на основе javadoc-комментариев, имеющихся в коде, создается документация к проекту в виде набора HTML-файлов с описаниями классов.

Программа javadoc в качестве входных данных использует исходный файл java-программы, анализирует javadoc-комментарии, расположенные перед классами, методами и конструкторами, и генерирует HTML-файл, являющийся описанием класса. Информация о каждом классе размещается в отдельном HTML-файле. Программа javadoc формирует также иерархическое дерево наследования классов и организует поиск по индексу.

Поскольку в результате генерации документации создается HTML-документация, в комментариях допускается применение тегов форматирования, таких как <b> и <p>. Однако теги заголовков <h1> … <h6> и тег <hr> использовать нельзя, так как они применяются javadoc для создания структуры документации.

Javadoc поддерживает специальные теги, начинающиеся с символа @. Подробное описание этих тегов можно найти в документации: (http://www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html).

Основные теги документации

Тег @autor

Тег @autor документирует автора класса и имеет следующий синтаксис:

```
@author описание
```

В описании указывается имя автора класса. Тег @autor можно использовать только к документации для класса.

```
/**
* Разъяснение содержания и особенностей класса...
* @author Имя Фамилия (автора)
* @version 1.0 (это версия)
*/
```

Тег @exception

Тег @exception описывает исключение для метода:

```
@exception exception-name описание
```

– exception-name – полное составное имя исключения;

– описание – строка, которая описывает, как может происходить исключение.

Тег @exception можно использовать только в документации метода.

Тег @throws

Тег @throws имеет такое же значение, как и @exception.

Тег {@link}

Тег {@link} обеспечивает встроенную гиперссылку на дополнительную информацию:

```
{@link name text}
```

– name – имя класса или метода, к которому добавляется гиперссылка;

– text – отображаемая строка.

Тег @param

Тег @param документирует параметр метода:

```
@param parameter-name описание
```

parameter-name определяет имя параметра метода. Значение этого параметра описывается в описании. Тег @param можно использовать только в документации для метода.

Тег @return

Тег @return описывает возвращаемое значение метода:

```
@return описание
```

В описании описывается тип и значение, возвращаемые методом. Тег @return можно использовать только в документации метода.

Тег @see

Тег @see обеспечивает ссылку к дополнительной информации. Он использует следующие формы:

– `@see anchor`

– `@see pkg.class#member text`

В первой форме anchor – гиперссылка к абсолютному или относительному URL-адресу.

Во второй форме pkg.class#member определяет имя элемента, а text – текст, отображаемый для этого элемента. Текстовый параметр необязательный. Имя члена также является необязательным. Таким образом, можно определить ссылку на пакет, класс или интерфейс в дополнение к ссылке на определенный метод или поле.

Например, можно использовать тег @see, чтобы сослаться на другой класс, поле или метод, или даже на другой Internet-ресурс.

```
/**
 * @see java.lang.String
 * @see java.lang.Math#PI
 * @see <a href="java.sun.com">Official Java site</a>
 */
```

Первая ссылка указывает на класс String (java.lang – название библиотеки, в которой находится этот класс), вторая – на поле PI класса Math (символ # разделяет название класса и его поля), третья – ссылается на официальный сайт Java.

Тег @code

Тег @code используется для форматирования ключевых слов. Например:

```
/**
 * Метод для определения местонахождения книги:
 * {@code false} - книга находится в библиотеке (свободна),
 * {@code true} - книга находится на руках
 * @return the status ({@code true} or {@code false})
 */
```

Тег @version

Тег @version определяет версию класса. Он имеет следующий синтаксис:

```
@version info
```

info – строка, которая содержит информацию о версии (обычно номер версии). Тег @version можно использовать только в документации для класса.

Команда Выполнить → Создать документацию Java (*Run → Generate Javadoc*) запускает создание документации по проекту. При этом из исходных кодов классов проекта выбирается информация, помеченная как javadoc-комментарии, на этой основе создается HTML-документ.

После генерации головная страница документации автоматически запускается в браузере. Пример документации класса Book приведен на рис. П2.1.

Для просмотра документации следует зайти в папку проекта, затем – в папку dist\javadoc и двойным щелчком запустить на просмотр файл index.html.

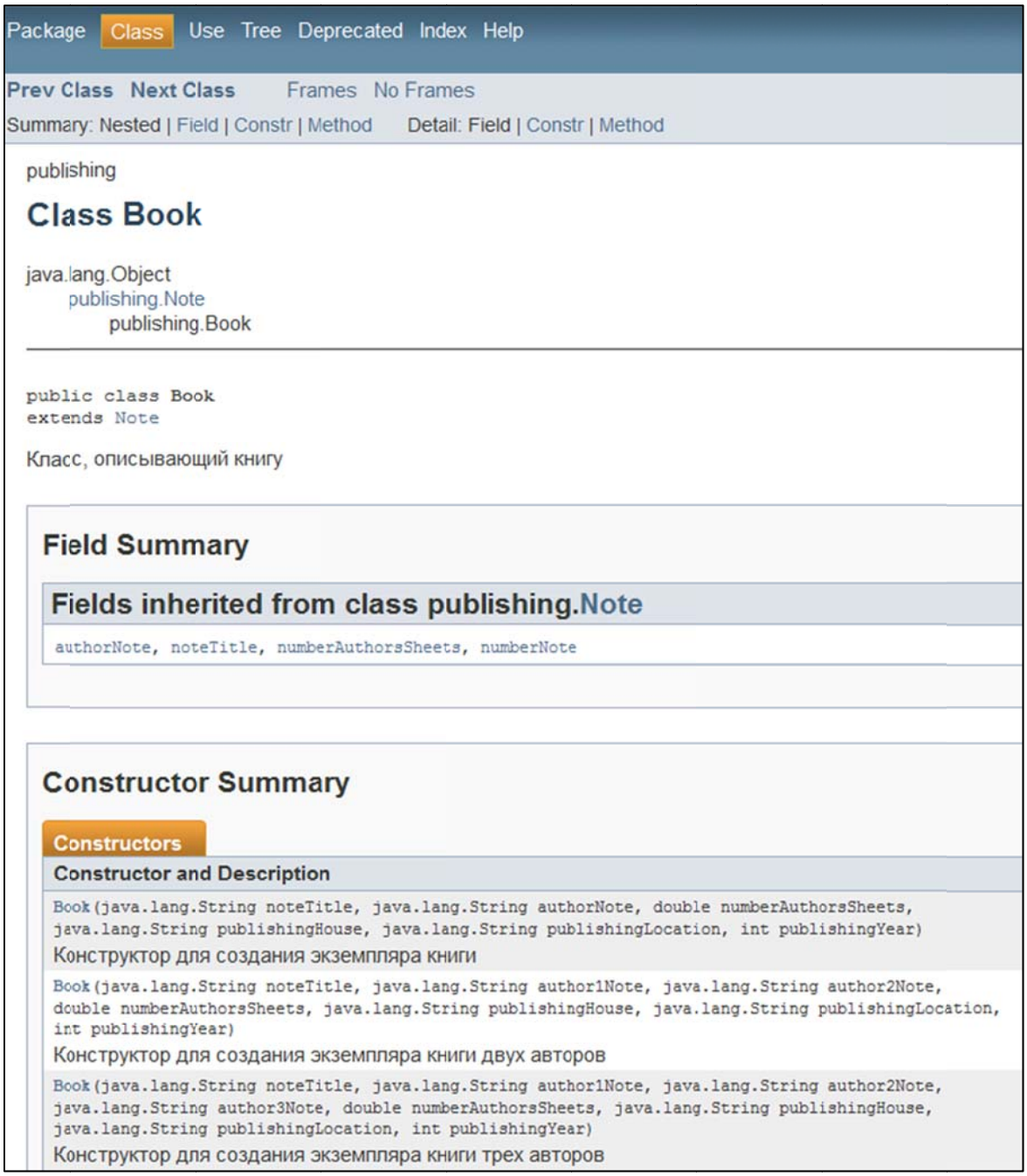

Рис. П2.1. Документация класса Book

Программный код классов проекта

Класс Author

```
package publishing;

/**
 * Класс, описывающий автора публикуемого материала
 * @author Болбот О.М.
 */
public class Author {
   private String surname; // фамилия
   private String name; // имя
   private String secondName; // отчество
   private String passportData; // паспортные данные
   private String address; // адрес
   private String phoneNumber; // контактный телефон

   /**
   * Конструктор для создания экземпляра автора
   * @param surname - фамилия
   * @param name - имя
   * @param secondName - отчество
   * @param passportData - паспортные данные
   * @param address - адрес
   * @param phoneNumber - контактный телефон
   */
 public Author(String surname, String name, String secondName,
       String passportData, String address, String phoneNumber) {
     this.surname = surname;
     this.name = name;
     this.secondName = secondName;
     this.passportData = passportData;
     this.address = address;
     this.phoneNumber = phoneNumber;
 }

 /**
 * Метод, формирующий полную информацию об авторе
 * @return the str - возвращает строку
 */
 @Override
 public String toString() {
   String str = surname + " " + name + " " + secondName + ", " +
```

```
            "паспортные данные: " + passportData + ", " +
            "\nадрес: " + address + ", конт. телефон: " + phoneNumber;
        return str;
    }

    /**
    * Метод, возвращающий фамилию и инициалы автора
    * @param surname - фамилия
    * @param name - имя
    * @param secondName - отчество
    * @return the str - возвращает строку типа Иванов И.И.
    */
    String getShortName(String surname, String name, String second-
Name) {
        String str = surname + " " + name.substring(0, 1) + "." +
                     secondName.substring(0, 1) + ".";
        return str;
    }

    /**
    * Перегруженный метод, возвращающий фамилию и инициалы автора
    * @param surname - фамилия
    * @param name - имя
    * @return the str - возвращает строку типа Иванов И.
    */
    String getShortName(String surname, String name) {
        String str = surname + " " + name.substring(0, 1) + ".";
        return str;
    }

    /**
    * Метод, возвращающий фамилию автора
    * @return the surname
    */
    public String getSurname() {
        return surname;
    }

    /**
    * Метод, возвращающий имя автора
    * @return the name
    */
    public String getName() {
        return name;
    }
```

```
    /**
     * Метод, возвращающий отыество автора
     * @return the secondName
     */
    public String getSecondName() {
        return secondName;
    }

    /**
     * Метод, возвращающий поспортные данные
     * @return the passportData
     */
    public String getPassportData() {
        return passportData;
    }

    /**
     * Метод, возвращающий адрес автора
     * @return the address
     */
    public String getAddress() {
        return address;
    }

    /**
     * Метод, возвращающий телефон автора
     * @return the phoneNumber
     */
    public String getPhoneNumber() {
        return phoneNumber;
    }

    /**
     * Метод, изменяющий фамилию автора
     * @param surname the surname to set
     */
    public void setSurname(String surname) {
        this.surname = surname;
    }

    /**
     * Метод, изменяющий паспортные данные
     * @param passportData the passportData to set
     */
    public void setPassportData(String passportData) {
        this.passportData = passportData;
    }
```

```
     /**
     * Метод, изменяющий адрес автора
     * @param address the address to set
     */
    public void setAddress(String address) {
         this.address = address;
     }

     /**
     * Метод, изменяющий телефон автора
     * @param phoneNumber the phoneNumber to set
     */
    public void setPhoneNumber(String phoneNumber) {
         this.phoneNumber = phoneNumber;
     }

    }
```

Класс Note

```
    package publishing;

    /**
     * Класс, описывающий публикуемый материал
     * @author Болбот О.М.
     */
    public class Note {
       protected int numberNote;              // порядковый номер ма-
териала
       private static int lastNumberNote = 0; // последний порядковый
номер
       protected String noteTitle;            // название мате-риала
       protected String authorNote;           // автор материала
       protected double numberAuthorsSheets;  // количество авторских
листов

     /**
     * Конструктор для создания экземпляра публикуемого материала
     * @param noteTitle - название материала
     * @param authorNote - автор материала
     * @param numberAuthorsSheets - количество авторских листов
     */
    public Note(String noteTitle, String authorNote,
                double numberAuthorsSheets) {
         numberNote = ++lastNumberNote; // порядковый номер материала
```

```
        this.noteTitle = noteTitle;
        this.authorNote = authorNote;
        this.numberAuthorsSheets = numberAuthorsSheets;
    }

    /**
    * Конструктор для создания экземпляра публикуемого материала
    * двух авторов
    * @param noteTitle - название материала
    * @param author1Note - 1 автор материала
    * @param author2Note - 2 автор материала
    * @param numberAuthorsSheets - количество авторских листов
    */
    public Note(String noteTitle, String author1Note, String au-
thor2Note,
                double numberAuthorsSheets) {
        numberNote = ++lastNumberNote; // порядковый номер материала
        this.noteTitle = noteTitle;
        this.authorNote = author1Note + ", " + author2Note;
        this.numberAuthorsSheets = numberAuthorsSheets;
    }

    /**
    * Конструктор для создания экземпляра публикуемого материала
    * трех авторов
    * @param noteTitle - название материала
    * @param author1Note - 1 автор материала
    * @param author2Note - 2 автор материала
    * @param author3Note - 3 автор материала
    * @param numberAuthorsSheets - количество авторских листов
    */
    public Note(String noteTitle, String author1Note, String au-
thor2Note, String author3Note, double numberAuthorsSheets) {
        // создаем порядковый номер материала
        numberNote = ++lastNumberNote;
        this.noteTitle = noteTitle;
        this.authorNote = author1Note + ", " + author2Note + ", "
                          + author3Note;
        this.numberAuthorsSheets = numberAuthorsSheets;
    }

    /**
    * Метод, формирующий информацию о публикуемом материале
    * @return the str - возвращает строку
    */
    @Override
    public String toString() {
```

```
        String str = numberNote + " " + authorNote + " " + noteTitle +
                     ", авт. листов: " + numberAuthorsSheets;
        return str;
     }

     /**
     * Метод для подсчета количества поступивших в издательство ма-
териалов
     * @return lastNumberNote
     */
    public static int getLastNumberNote() {
         return lastNumberNote;
     }

     /**
     * Метод чтения порядкового номера материала
     * @return numberNote
     */
    public int getNumberNote() {
         return numberNote;
     }

     /**
     * Метод чтения названия материала
     * @return noteTitle
     */
    public String getNoteTitle() {
         return noteTitle;
     }

     /**
     * Метод чтения автора материала
     * @return authorNote
     */
    public String getAuthorNote() {
         return authorNote;
     }

     /**
     * Метод чтения количества авторских листов
     * @return numberAuthorsSheets
     */
    public double getNumberAuthorsSheets() {
         return numberAuthorsSheets;
     }
```

```
     /**
     * Метод изменения названия материала
     * @param noteTitle - новое название
     */
    public void setNoteTitle(String noteTitle) {
         this.noteTitle = noteTitle;
     }

     /**
     * Метод изменения количества авторских листов
     * @param numberAuthorsSheets - новое количество авторских ли-
стов
     */
    public void setNumberAuthorsSheets(double numberAuthorsSheets) {
         this.numberAuthorsSheets = numberAuthorsSheets;
     }

     /**
     * Метод подсчета количества страниц
     * @return int - Кол-во страниц
     */
    public int getNumberPages() {
         return (int) Math.round(numberAuthorsSheets * 40000 / 3700);
     }

    }
```

Класс Book

```
    package publishing;

    /**
     * Класс, описывающий книгу
     * @author Болбот О.М.
     */
    public class Book extends Note{
         private String publishingHouse;        // Издательство
         private String publishingLocation;     // Город
         private int publishingYear;            // Год издания
         private int numberPage;                // Кол-во страниц

    /**
     * Конструктор для создания экземпляра книги
     * @param noteTitle - название
     * @param authorNote - автор
     * @param numberAuthorsSheets - количество авторских листов
     * @param publishingHouse - издательство
```

```
     * @param publishingLocation - город
     * @param publishingYear - год издания
     */
    public Book(String noteTitle, String authorNote,
            double numberAuthorsSheets, String publishingHouse,
            String publishingLocation, int publishingYear) {
        super(noteTitle, authorNote, numberAuthorsSheets);
        this.publishingHouse = publishingHouse;
        this.publishingLocation = publishingLocation;
        this.publishingYear = publishingYear;
        numberPage = getNumberPages();
     }

    /**
     * Конструктор для создания экземпляра книги
     * двух авторов
     * @param noteTitle - название
     * @param author1Note - автор 1
     * @param author2Note - автор 2
     * @param numberAuthorsSheets - количество авторских листов
     * @param publishingHouse - издательство
     * @param publishingLocation - город
     * @param publishingYear - год издания
     */
    public Book(String noteTitle, String author1Note, String au-
thor2Note,
            double numberAuthorsSheets, String publishingHouse,
            String publishingLocation, int publishingYear) {
        super(noteTitle, author1Note, author2Note, numberAuthors-
Sheets);
        this.publishingHouse = publishingHouse;
        this.publishingLocation = publishingLocation;
        this.publishingYear = publishingYear;
        numberPage = getNumberPages();
     }

    /**
     * Конструктор для создания экземпляра книги
     * трех авторов
     * @param noteTitle - название
     * @param author1Note - автор 1
     * @param author2Note - автор 2
     * @param author3Note - автор 3
     * @param numberAuthorsSheets - количество авторских листов
     * @param publishingHouse - издательство
     * @param publishingLocation - город
```

```
     * @param publishingYear - год издания
     */
    public Book(String noteTitle, String author1Note, String au-
thor2Note,
            String author3Note, double numberAuthorsSheets,
            String publishingHouse, String publishingLocation,
            int publishingYear) {
        super(noteTitle, author1Note, author2Note, author3Note,
                numberAuthorsSheets);
        this.publishingHouse = publishingHouse;
        this.publishingLocation = publishingLocation;
        this.publishingYear = publishingYear;
        numberPage = getNumberPages();
     }

    /**
     * Метод, формирующий информацию о книге
     * @return the str - возвращает строку
     */
    @Override
    public String toString() {
       String str = numberNote + " " + authorNote + " " + noteTitle +
            ". - " + publishingLocation + ".: " + publishingHouse +
            ", " + publishingYear + ".- " + numberPage + " с.";
       return str;
     }

    /**
     * Метод для чтения названия издательства
     * @return publishingHouse
     */
    public String getPublishingHouse() {
        return publishingHouse;
     }

    /**
     * Метод для чтения места издания (города)
     * @return publishingLocation
     */
    public String getPublishingLocation() {
        return publishingLocation;
     }

    /**
     * Метод для чтения года издания
     * @return publishingYear
     */
```

```
public int getPublishingYear() {
     return publishingYear;
 }

/**
 * Метод для изменения года издания
 * @param publishingYear
 */
public void setPublishingYear(int publishingYear) {
     this.publishingYear = publishingYear;
 }

}
```

Класс Magazine

```
package publishing;

/**
 * Класс, описывающий журнал
 * @author Болбот О.М.
 */
public class Magazine {
 private String magazineTitle;          // название журнала
 private int numberMagazine;            // номер выпуска
 private int publishingYear;            // Год издания
 private Note notes [];                 // массив статей
 private String publishingHouse;        // Издательство
 private String publishingLocation;     // Город
 private int numberPage;                // Кол-во страниц
 private Contents contents;             // Содержание журнала

/**
 * Конструктор для создания экземпляра журнала
 * @param magazineTitle - название
 * @param numberMagazine - номер выпуска
 * @param publishingYear - год издания
 * @param notes - массив статей
 * @param publishingHouse - издательство
 * @param publishingLocation - город
 */
public Magazine(String magazineTitle, int numberMagazine,
        int publishingYear, Note[] notes, String publishingHouse,
        String publishingLocation) {
     this.magazineTitle = magazineTitle;
     this.numberMagazine = numberMagazine;
```

```
        this.publishingYear = publishingYear;
        this.notes = notes;
        this.publishingHouse = publishingHouse;
        this.publishingLocation = publishingLocation;
        // Формирование оглавления
        contents = new Contents(notes);
    }

    /**
     * Метод, формирующий информацию о журнале
     * @return str - возвращает строку
     */
    @Override
    public String toString() {
        String str = "Журнал " + magazineTitle + " № " + number-
Magazine +
                     " " + publishingYear + ". - " + publishingLo-
cation + ".: "
                     + publishingHouse;
        str += contents.toString();
        return str;
    }

    /**
     * Метод для чтения названия журнала
     * @return magazineTitle
     */
    public String getMagazineTitle() {
        return magazineTitle;
    }

    /**
     * Метод для чтения номера журнала
     * @return numberMagazine
     */
    public int getNumberMagazine() {
        return numberMagazine;
    }

    /**
     * Метод для чтения года издания
     * @return publishingYear
     */
    public int getPublishingYear() {
        return publishingYear;
    }
```

```
    /**
     * Метод для получения массива статей журнала
     * @return notes
     */
    public Note[] getNotes() {
         return notes;
     }

    /**
     * Метод для изменения массива статей журнала
     * @param notes the notes to set
     */
    public void setNotes(Note[] notes) {
         this.notes = notes;
         contents.createContents(notes);
     }

    /**
     * Метод для чтения названия издательства
     * @return publishingHouse
     */
    public String getPublishingHouse() {
         return publishingHouse;
     }

    /**
     * Метод для чтения места издания (города)
     * @return publishingLocation
     */
    public String getPublishingLocation() {
         return publishingLocation;
     }

    /**
     * Метод, вычисляющий количество страниц в журнале
     * @return numberPage
     */
    public int getNumberPage() {
         numberPage = contents.getNumberPages();
         return numberPage;
    }
}
```

Класс Contents

```
package publishing;

/**
 * Класс, формирующий содержание журнала
 * @author Болбот О.М.
 */
public class Contents {
     private Note notes [];         // массив статей
     // двумерный массив номеров страниц (оглавление)
     private int pageNumberNote [] [];
     private int numberPage;        // текущий номер страницы

/**
 * Конструктор для создания содержания журнала
 * @param notes - массив статей
 */
public Contents(Note[] notes) {
       this.notes = notes;
       // Массив pageNumberNote содержит для каждого материала:
       // 1. Номер начальной страницы материала
       // 2. Количество страниц, необходимых для размещения ма-
териала
       pageNumberNote = new int [notes.length] [2];
       createContents(notes);
 }

/**
 * Метод формирования оглавления
 */
public void createContents(Note[] notes){
     int i;
     // Очистка оглавления
     for (i = 0; i < notes.length; i++){
         pageNumberNote[i][0] = 0;
         pageNumberNote[i][1] = 0;
     }
     // первая страница - титульная
     // вторая страница содержит оглавление
     // материалы начинаются с третьей страницы
     numberPage = 3;
     i = 0;
     while (notes[i] != null){
         pageNumberNote[i][0] = numberPage; // начальный номер
         // объем материала
         pageNumberNote[i][1] = notes [i].getNumberPages();
         // корректировка текущего номера страниц
```

```
            numberPage += pageNumberNote[i][1];
            i++;
        }
    }

    /**
     * Метод подсчета общего количества страниц в журнале
     * @return int - Кол-во страниц
     */
    public int getNumberPages() {
        return numberPage - 1;
     }

    /**
     * Метод, выводящий оглавление
     * @return the str - возвращает строку
     */
    @Override
    public String toString() {
        String str = "\nСодержание:";
        int i = 0;
        while (notes[i] != null){
            str += "\n" + (i+1) + " " + notes[i].getAuthorNote() +
" " +
                    notes[i].getNoteTitle() + "\t" + pageNumber-
Note[i][0];
            i++;
        }
        return str;
     }

}
```

Класс Publishing

```
package publishing;

/**
 * Класс "Издательство"
 * @author Болбот О.М.
 */
public class Publishing {
    private static final String PUBLISHING_HOUSE =
"Эрудит";//Издательство
    private static final String PUBLISHING_LOCATION = "Мн"; //
Город Минск
    public static void main(String[] args) {
```

```
                // Создание автора
                System.out.println("Авторы:");
                Author vasilev = new Author("Васильев", "Павел",
"Иванович",
                "MP2035648, выдан 18.05.2011",
                "г. Минск, ул. Васнецова, д. 45, кв. 79",
                "8(029)33-564-78-02");
        System.out.println(vasilev.toString());
        String shortNameVasilev = vasilev.getShortName(vasilev.
getSurname(),
                vasilev.getName(), vasilev.getSecondName());
        System.out.println("Фамилия и инициалы: " + shortNameVasilev);
         // vasilev.setPhoneNumber("8(029)44-564-78-02");
         // System.out.println(vasilev.toString());

        Author petrov = new Author("Петров", "Виталий", "Сергеевич",
                "AH5601285, выдан 21.08.2015",
                "г. Минск, ул. Сурганова, д. 23, кв. 20",
                "8(029)66-125-70-98");
        System.out.println(petrov.toString());

        Author vilson = new Author("Вилсон", "Альгис", "",
                "MP4503878, выдан 02.11.2001",
                "г. Минск, ул. Плеханова, д. 112, кв. 125",
                "8(029)44-854-05-91");
        System.out.println(vilson.toString());
        String shortNameVilson = vilson.getShortName(vilson. get-
Surname(),
                vilson.getName());
        System.out.println("Фамилия и инициалы: " + shortNameVilson);

        // Создание материала
        System.out.println("\nПубликуемые материалы:");
        Note noteVasilev1 = new Note("Создание классов в Java",
                shortNameVasilev, 3.6);
        Note noteVasilev2 = new Note("Среда NetBeans",
                shortNameVasilev, 2.8);
        System.out.println(noteVasilev1.toString());
        System.out.println("Кол-во страниц " + noteVa-
silev1.getNumberPages());
        System.out.println(noteVasilev2.toString());
        System.out.println("Кол-во страниц " + noteVasilev2.get-
NumberPages());

       // noteVasilev1.setNoteTitle("Классы в Java");
       // noteVasilev1.setNumberAuthorsSheets(3.5);
       // System.out.println(noteVasilev1.toString());
```

```
      // Создание книг
         System.out.println("\nКниги:");
         Book bookVasilev3 = new Book("Программирование в Java",
                 shortNameVasilev, 30.5, PUBLISHING_HOUSE,
                 PUBLISHING_LOCATION, 2016);
         System.out.println(bookVasilev3.toString());

         String  shortNamePetrov  =  petrov.getShortName(petrov.get-
Surname(),
                 petrov.getName(), petrov.getSecondName());
         Book bookPetrov4 = new Book("Программирование на VB",
                 shortNamePetrov, 22.5, PUBLISHING_HOUSE,
                 PUBLISHING_LOCATION, 2016);
         System.out.println(bookPetrov4.toString());

         // Создание журнала
         System.out.println("\nЖурналы:");
         Note notes [] = new Note [10]; // массив статей
         notes[0] = noteVasilev1;
         notes[1] = noteVasilev2;
         Magazine magazine = new Magazine("Программирование", 1, 2016,
                 notes, PUBLISHING_HOUSE, PUBLISHING_LOCATION);
         System.out.println(magazine.toString());
         System.out.println("Всего    страниц:    "    +    maga-
zine.getNumberPage());

         // Перегрузка конструкторов
         System.out.println("\nСтатья двух авторов:");
         Note noteVasilev3 = new Note("Реализация принципов ООП",
         shortNameVasilev, shortNamePetrov, 2.4);
         System.out.println(noteVasilev3.toString());
         // Добавим ее в журнал
                 notes[2] = noteVasilev3;
         magazine.setNotes(notes);
         System.out.println(magazine.toString());
         System.out.println("Всего    страниц:    "    +    maga-
zine.getNumberPage());

         System.out.println("\nКнига трех авторов:");
         Book book5 = new Book("Современные языки программирования",
                 shortNameVasilev, shortNameVilson, shortNamePetrov,
                 22.5, PUBLISHING_HOUSE, PUBLISHING_LOCATION, 2016);
         System.out.println(book5.toString());
     }

}
```

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Аккуратов, Е. Е. Знакомьтесь: Java : самоучитель / Е. Е. Аккуратов. – ИД «Вильямс», 2005. – 230 с.

2. Васильев, А. Н. Java. Объектно-ориентированное программирование : учебное пособие / А. Н. Васильев. – СПб. : Питер, 2013. – 400 с.

3. Монахов, В. В. Язык программирования Java и среда NetBeans / В. В. Монахов. – 3-е изд., перераб. и доп. – СПб. : БХВ-Петербург, 2011. – 704 с.

4. Хабибуллин, И. Ш. Java 7 : И. Ш. Хабибуллин. – СПб. : БХВ-Петербург, 2012. – 768 с.

5. Хорстманн, К. С. Библиотека профессионала. Java 2 : пер. с англ. / К. С. Хорстманн, Г. Корнелл. – М.: ИД «Вильямс», 2003. – 848 с.

6. Шилдт, Г. Java. Полное руководство : пер. с англ. / Г. Шилдт. – 8-е изд. – М. : ИД «Вильямс», 2012. – 1104 с.

7. Эккель, Б. Философия Java. Библиотека программиста / Б. Эккель. – СПб. : Питер, 2009. – 640 с.

8. Liang, Y. Introduction to Java programming / Y. Liang. – New Jersey : Pearson Higher Education, 2011. – 756 с.

Учебное издание

БОЛБОТ Ольга Михайловна
СИДОРИК Валерий Владимирович

КЛАССЫ В ЯЗЫКЕ ПРОГРАММИРОВАНИЯ JAVA

Учебно-методическое пособие
для студентов и слушателей системы повышения квалификации
и переподготовки

Под общей редакцией *В. В. Сидорика*

Редактор *А. С. Кириллова*
Компьютерная верстка *Е. А. Беспанской*

Подписано в печать 20.01.2020. Формат 60×84 $^{1}/_{8}$. Бумага офсетная. Ризография.
Усл. печ. л. 8,95. Уч.-изд. л. 3,50. Тираж 100. Заказ 836.

Издатель и полиграфическое исполнение: Белорусский национальный технический университет.
Свидетельство о государственной регистрации издателя, изготовителя, распространителя
печатных изданий № 1/173 от 12.02.2014. Пр. Независимости, 65. 220013, г. Минск.