



*Саратовский госуниверситет*



**Механико-математический факультет**

- каф. математического и компьютерного моделирования ауд. 302, 301, 220, 64 III кор.
- раб. тел. 51-84-80
- email: evilgraywolf@gmail.com

## Содержание

<b>1</b>	<b>Введение в Python</b>	<b>2</b>
1.1	Hello, world! . . . . .	2
1.2	Переменные . . . . .	4
1.3	Оператор присваивания . . . . .	4
1.4	Основные арифметические операции . . . . .	5
1.5	Основные логические операции . . . . .	8
<b>2</b>	<b>Основные понятия</b>	<b>8</b>
<b>3</b>	<b>Задание 1</b>	<b>16</b>
<b>4</b>	<b>Задание 2</b>	<b>29</b>
<b>5</b>	<b>Задание 3</b>	<b>40</b>
<b>6</b>	<b>Задание 4</b>	<b>43</b>
6.1	Виджеты для работы с датами . . . . .	70
6.2	Использование MVC . . . . .	72
<b>7</b>	<b>Задание 5</b>	<b>86</b>

# 1 Введение в Python

Учебное пособие рассчитано на студентов, имеющих опыт программирования на императивных языках программирования. Императивное программирование - вид программирования при котором программа представляет набор последовательно выполняемых команд. Для языков программирования, ориентированных на такую парадигму характерно наличие переменных, операторов присваивания, ветвления и циклов. В данной главе будут рассмотрены особенности использования этих конструкций в языке Python.

## 1.1 Hello, world!

Чтобы начать программировать на Python создайте текстовый файл. Желательно его преименовать, задав расширение ".py" в этом случае текстовые редакторы, имеющие функцию синтаксической подсветки (например Kate в Linux или Context в Windows) опознают созданный файл и включают подсветку автоматически. Откройте созданный файл с помощью текстового редактора и введите следующий текст:

```
1 | print '*****'  
2 | print '* Hello, world! *'  
3 | print '*****'
```

Затем, сохраните изменения и в консоли (удобно использовать текстовый редактор со встроенной консолью) введите "python затем через пробел имя файла программы.

```
1 | python ex1.py
```

Нажмите Enter. В консоли появится результат работы программы. Функция print в данном случае выводит на экран строку символов заключенную в апострофы или кавычки.

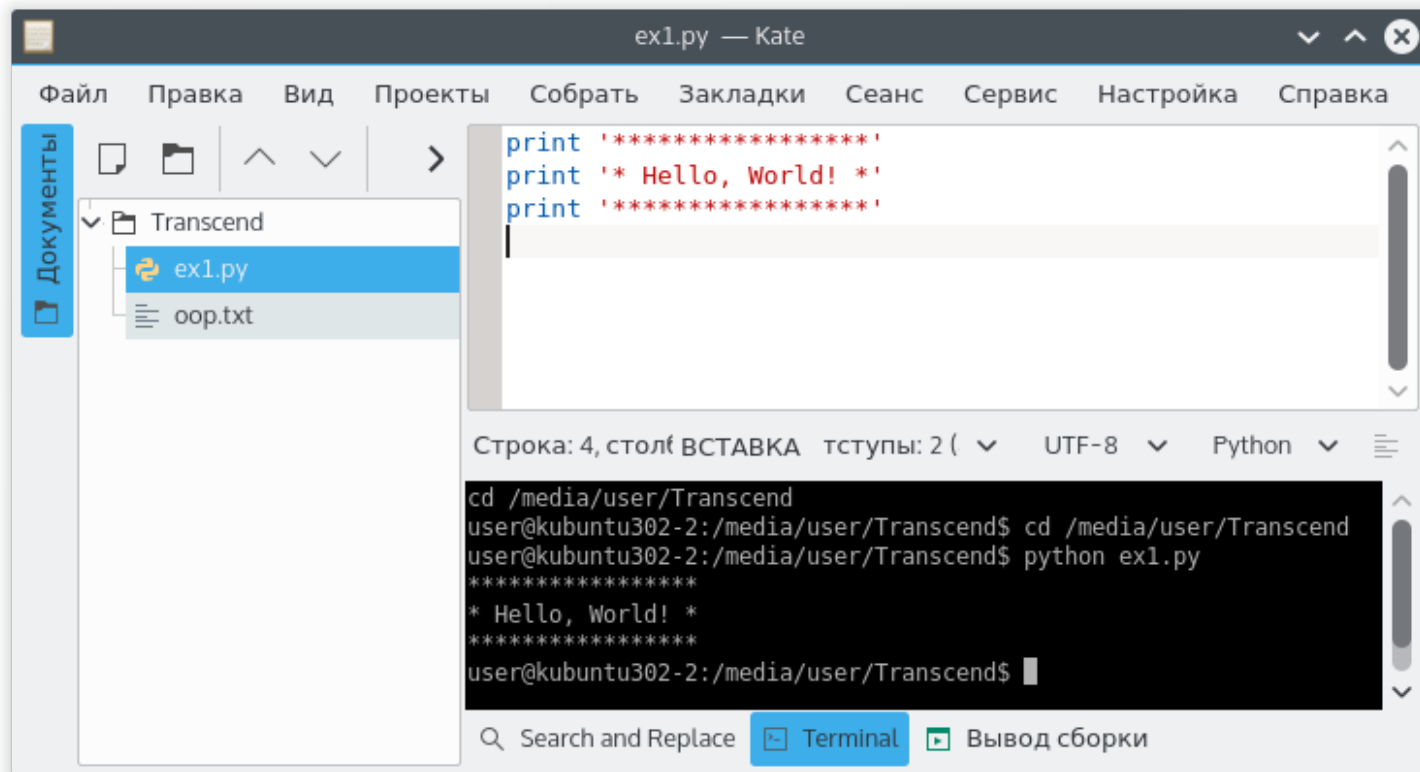


Рис. 1: Первая программа

Если в программе используются символы кириллицы, например в строковых переменных или комментариях, в начале программы следует указать кодировку, используемую в файле программы, например так:

```

1 | #-*- coding:utf-8 -*-
2 | print '*****'
3 | print '* Здравствуй, мир! *'
4 | print '*****'

```

## 1.2 Переменные

К именам переменных в Python применяются те же требования, что и в большинстве других языков программирования - имя переменной может состоять из букв латинского алфавита, цифр, символа подчеркивания. Имя переменной не должно начинаться с цифры, содержать пробелы, знаки препинания, знаки математических операций. Заглавные и строчные буквы в идентификаторах Python различаются (например **None** не то же самое, что **none**).

## 1.3 Оператор присваивания

Оператор присваивания обозначается символом `=`. Например:

```
1 | n=1
2 | print n
```

Результат работы программы:

```
1 | 1
```

В языке Python есть возможность присвоить сразу нескольким переменным несколько значений:

```
1 | x,y,color=10,15,'white'
2 | print x,y,color
```

Результат:

```
1 | 10 15 white
```

При этом количество и порядок перечисления переменных и значений должны совпадать.

```
1 | x,y,z,color=10,15,'white'
2 | print x,y,z,color
```

Результат - сообщение об ошибке:

```

1 | Traceback (most recent call last):
2 |   File "ex.py", line 1, in <module>
3 |     x,y,z,color=10,15,'white'
4 | ValueError: need more than 3 values to unpack

```

Переменные могут быть различных типов, основные из которых целые и действительные числа, строки. Тип переменной задается при первом присвоении ей значения. Целые числа задаются цифрами.

```

1 | i1=3
2 | i2=-5

```

Действительные числа задаются цифрами с десятичным разделителем точкой.

```

1 | r1=3.14
2 | r2=-5.0

```

Строки задаются любыми последовательностями символов, заключенными в кавычки или апострофы, при этом, если строка заключена в кавычки, внутри нее можно использовать апострофы и наоборот.

```

1 | s1='python'
2 | s2="python"
3 | s3='Тема доклада: "Роль этого в жизни каждого"'
4 | s4="Жанна д'Арк (1412-1431)"

```

## 1.4 Основные арифметические операции

К основным арифметическим операциям относится сложение «+», вычитание «-», умножение «\*», деление «/». Эти операции применимы к переменным и константам.

```

1 | a,b=3,5
2 | c=a+b
3 | print c
4 | print 3-8
5 | a+=3

```

```

6 | print a
7 | b=b+1
8 | print b
9 | print 3*4
10 | print 30/6
11 | print (a+b)*c

```

Результат работы программы

```

1 | 8
2 | -5
3 | 6
4 | 6
5 | 12
6 | 5
7 | 96

```

Применение основных арифметических операций в python2 имеет ряд особенностей, например в результате деления целого числа на целое число получается целое число, как при делении с остатком. Получить остаток от деления в этом случае можно с помощью операции «%».

```

1 | print 34/10
2 | print 34%10

```

Результат:

```

1 | 3
2 | 4

```

Чтобы поделить целое число на целое число без остатка, нужно делимое умножить на 1.0. Если умножить делитель на 1.0, то умножение нужно взять в скобки, иначе по порядку сначала произойдет деление с остатком, а потом результат будет умножен на 1.0:

```

1 | print 34*1.0/10
2 | print 34/10*1.0
3 | print 34/(10*1.0)

```

Результат:

```

1 | 3.4
2 | 3.0
3 | 3.4

```

В python3 операция «/» означает обычное деление, деление с остатком «//», остаток от деления, как и в python2 «%»:

```

1 | print 34/10
2 | print 34//10
3 | print 34%10

```

Результат:

```

1 | 3.4
2 | 3
3 | 4

```

К строкам применимы операции сложения со строкой и умножения на число:

```

1 | s1='one'
2 | s2='two'
3 | print s1+s2
4 | print s1+' '+s2
5 | print s1*3
6 | print '='*10

```

Результат:

```

1 | onetwo
2 | one two
3 | oneoneone
4 | =====

```



## 1.5 Основные логические операции

К основным логическим операциям относятся операции отрицания, сравнения констант и переменных ( $=$ ,  $<$ ,  $>$ ), проверка принадлежности элемента множеству, а также логические функции.

## 2 Основные понятия

**Объектно-ориентированное программирование** - методология программирования, основанная на представлении программы в виде совокупности **объектов**.

**Объект** - структура, обладающая **состоянием** и **поведением**.

**Состояние** объекта определяется набором значений его **атрибутов**.

**Атрибуты** объекта представляются набором переменных внутри данного объекта.

**Поведение** объекта определяется набором функций, называемых **методами** внутри данного объекта.

Структура объекта, то есть набор переменных и функций, определяется при описании **класса**. **Класс** является шаблоном по которому создаются однотипные объекты.

При создании объекта выполняется специальная функция, называемая **конструктор**. Основное ее назначение - выделение места в памяти под данный объект. Она возвращает адрес, по которому объект располагается в динамической памяти. Кроме того, конструктор может выполнять некоторые дополнительные действия, необходимые при создании объекта, например задание начальных значений атрибутов.

**Инкапсуляция** - свойство объекта скрывать свою структуру внутри себя, представляясь пользователям «черным ящиком», предоставляя доступ лишь к некоторым своим функциям.

Атрибуты и методы определенного объекта, доступ к которым разрешен только из методов данного объекта называются закрытыми (**Private**).

Методы определенного объекта, через которые осуществляется доступ к функциям данного объекта из вне, называются открытыми (**Public**).

**Наследование** - создание нового класса (**наследника**) на базе одного или нескольких уже существующих классов (**предков**). Если наследование происходит от нескольких классов, то оно называется множественным. При наследовании, при определении класса указываются классы, от которых происходит наследование, при этом класс-наследник будет обладать всеми атрибутами и методами классов-предков. При этом наследуемые атрибуты и методы можно переопределить, а

также определить новые. Явление наследования классов также называют обобщением (**Generalization**). При этом в классах, составляющих программу, определяются одинаковые атрибуты и методы и помещаются в общий класс-предок.

Закрытые атрибуты и методы класса предка недоступны в классе потомке. Существуют защищенные (**Protected**) атрибуты и методы, которые доступны в классе потомке, но при этом недоступны из остальных частей программы.

**Полиморфизм** - изменение поведения класса потомка относительно классу предку при сохранении сигнатуры методов.

**Агрегация** - включение одного объекта в другой.

Поясним перечисленные определения и явления на примере.

Опишем класс, описывающий некоторые предметы (item), имеющие вес. Соответственно класс будет иметь атрибут вес (weight).

```
1  -*- coding:utf-8 -*-
2  class item:
3      def __init__(self,weight=0):
4          self.setWeigth(weight)
5      def getWeigth(self):return self.__weight
6      def setWeigth(self,value):self.__weight=value
```

В строке 2 данного листинга с помощью ключевого слова «class» задается имя класса. Далее идет описание его структуры. В строках 3 и 4 описывается конструктор класса. В Python он всегда называется «\_\_init\_\_». В скобках после имени функции перечисляются аргументы, при этом можно задать значения по умолчанию. В данном случае вес по умолчанию принимает значение 0. Если функция является методом класса, то первым аргументом всегда является ссылка на объект. Таким образом объект сможет обращаться к своим атрибутам и методам, как показано в строке 3. В данном случае устанавливается значение атрибута weight. В соответствии с принципом инкапсуляции прямой доступ к атрибутам объектов извне запрещен, поэтому атрибут описан как приватный (\_\_weight). В Python двойное подчеркивание в начале названия атрибута или метода означает приватность. Проверим, как это работает. Добавим в код следующие строки и запустим программу.

```
7  i=item(5)
8  print i.getWeigth()
9  print i.__weight
```

В результате выполнения программы будет выведено на экран следующее:

```

5
Traceback (most recent call last):
  File "item.py", line 45, in <module>
    print i.__weighth
AttributeError: item instance has no attribute '__weighth'

```

Как видим, строка 8 с обращением через публик-метод сработала нормально, а попытка обратиться к приватному атрибуту напрямую вызвала ошибку.

Теперь уберем строки 7-9 и опишем еще два класса, унаследовав их от item.

```

1  -*- coding:utf-8 -*-
2  class item:
3      def __init__(self,weighth=0):
4          self.setWeigth(weighth)
5      def getWeigth(self):return self.__weighth
6      def setWeigth(self,value):self.__weighth=value
7
8  class itemgroup(item):
9      def __init__(self,weighth=0,num=0):
10         item.__init__(self,weighth)
11         self.setNum(num)
12     def getNum(self):return self.__num
13     def setNum(self,value):self.__num=value
14     def getWeigth(self):return item.getWeigth(self)*self.getNum()
15
16 class container(item):
17     def __init__(self,weighth=0):
18         item.__init__(self,weighth)
19         self.__itemList=[]
20     def appendItem(self,item):self.__itemList.append(item)
21     def getWeigth(self):
22         weighth=item.getWeigth(self)
23         for i in self.__itemList:
24             weighth+=i.getWeigth()
25         return weighth
26 c=container()

```

```

27 i1=item(10)
28 i2=itemgroup(3,5)
29 c.appendItem(i1)
30 c.appendItem(i2)
31 print c.getWeigth()

```

В строке 8 определяем класс «itemgroup» описывающий сгруппированные однотипные предметы, например одинаковые товары в строке кассового чека - такая строка имеет наименование товара, цену, одинаковую для всех товаров и их количество. В нашем случае класс «itemgroup» будет обладать двумя атрибутами весом (weigth) и количеством (num). При этом вес и методы доступа к нему будут унаследованы от класса-предка «item». Метод «getWeigth» нужно переопределить, т. к. теперь он рассчитывается другим способом - вес одного предмета умножается на количество. При этом сигнатура метода остается неизменной. Это является примером полиморфизма.

Аналогичным образом определим класс «container» - предмет, обладающий весом и способный включать в себя другие предметы. Кроме веса, унаследованного от класса «item» - он содержит атрибут «\_\_itemList» - список включенных в него объектов (агрегация). Расчет веса предмета, как и в первом случае изменится - вес всего контейнера будет состоять из его собственного веса и весов всех включенных в него предметов, которые могут быть как простыми предметами, так и группами предметов и контейнерами, метод «getWeigth» от этого не изменится.

Результатом выполнения данной программы будет вывод на экран веса контейнера с содержащимися в нем предметами, в данном случае 25.

Теперь, если мы хотим добавить во все описанные классы общий атрибут, например название, достаточно добавить его в класс, являющийся общим предком.

```

1 #-*- coding:utf-8 -*-
2 class item:
3     def __init__(self,name='',weigth=0):
4         self.setName(name)
5         self.setWeigth(weigth)
6     def getName(self):return self.__name
7     def setName(self,value):self.__name=value
8     def getWeigth(self):return self.__weigth
9     def setWeigth(self,value):self.__weigth=value
10
11 class itemgroup(item):

```

```

12 def __init__(self,name='',weigh=0,num=0):
13     item.__init__(self,name,weigh)
14     self.setNum(num)
15 def getNum(self):return self.__num
16 def setNum(self,value):self.__num=value
17 def getWeigth(self):return item.getWeigth(self)*self.getNum()
18
19 class container(item):
20     def __init__(self,name='',weigh=0):
21         item.__init__(self,name,weigh)
22         self.__itemList=[]
23     def appendItem(self,item):self.__itemList.append(item)
24     def removeItem(self,name):
25         for i in self.__itemList:
26             if i.getName()==name:
27                 self.__itemList.remove(i)
28                 return i
29     def getWeigth(self):
30         weigh=item.getWeigth(self)
31         for i in self.__itemList:
32             weigh+=i.getWeigth()
33         return weigh
34
35 c=container()
36 i1=item(name='item1',weigh=10)
37 i2=itemgroup(name='item2',weigh=3,num=5)
38 c.appendItem(i1)
39 c.appendItem(i2)
40 print c.getWeigth()
41 c.removeItem('item1')
42 print c.getWeigth()

```

Если мы хотим, чтобы новый атрибут «name» инициализировался в конструкторе (что не обязательно), нужно внести соответствующие изменения в конструкторы всех классов. Теперь можно определить метод удаления предмета из контейнера по названию (см. строки 24-28). В результате работы этой программы будет выведен на экран вес контейнера с помещенными в него предметами, затем вес контейнера после удаления одного из предметов, в данном случае 25 и 15.

Однако, если класс «item» уже используется в работающей программе, то вносить изменения в него не рекомендуется, так как это может отразиться на использующем его коде, поэтому лучше поступить следующим образом:

```
1 class named(item):
2     def __init__(self,name=''):
3         self.setName(name)
4     def getName(self):return self.__name
5     def setName(self,value):self.__name=value
6
7 class nameditem(named,item):
8     def __init__(self,name='',weigh=0):
9         named.__init__(self,name)
10        item.__init__(self,weigh)
11
12 class nameditemgroup(itemgroup,nameditem):
13     def __init__(self,name='',weigh=0,num=0):
14         named.__init__(self,name)
15         itemgroup.__init__(self,weigh,num)
16
17 class namedcontainer(container,nameditem):
18     def __init__(self,name='',weigh=0):
19         named.__init__(self,name)
20         container.__init__(self,weigh)
21
22 c=namedcontainer('container1',20)
23 i1=nameditem('item1',10)
24 i2=nameditemgroup('itemgroup1',3,5)
25 c.appendItem(i1)
26 c.appendItem(i2)
27 print c.getName()
28 print i1.getName()
29 print i2.getName()
30 print c.getWeigth()
```

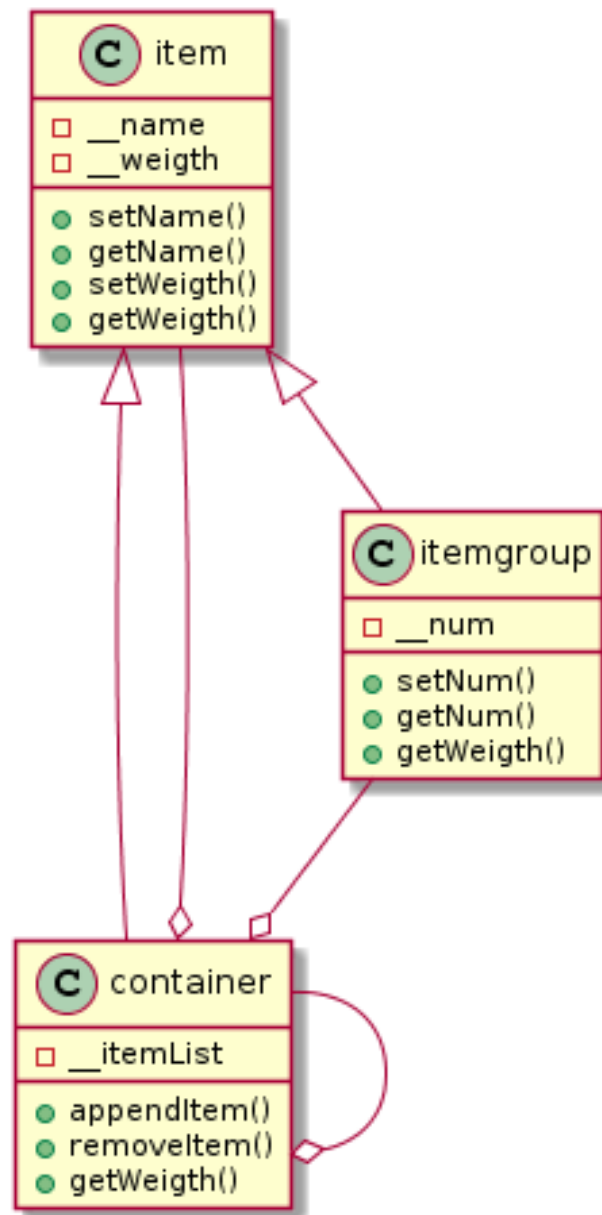


Рис. 2: Диаграмма классов

Связи между классами удобно представлять на диаграммах классов языка UML. Удобным ресурсом для создания таких диаграмм является PlantUML (<http://www.plantuml.com/plantuml/>). Ниже приведен скрипт, использовавшийся для создания диаграммы на рис. 2.

```
1 class item {
2   -__name
3   -__weight
4   +setName()
5   +getName()
6   +setWeight()
7   +getWeight()
8 }
9 class itemgroup {
10  -__num
11  +setNum()
12  +getNum()
13  +getWeight()
14 }
15 class container {
16  -__itemList
17  +appendItem()
18  +removeItem()
19  +getWeight()
20 }
21 item <|-- itemgroup
22 item <|-- container
23 container o-- item
24 container o-- itemgroup
25 container o-- container
```



### 3 Задание 1

В выбранной предметной области определить 2-3 связанных между собой сущности, их атрибуты. На языке Python описать классы, соответствующие найденным сущностям. Создать объекты описанной связки классов. Реализовать запросы, рекомендованные преподавателем. Для полученной программы построить диаграмму классов UML.

#### Пример выполнения

В качестве предметной области выбрана библиотека. Необходимо хранить информацию о книгах, авторах и издательствах. В рассматриваемой предметной области выделим следующие классы:

- автор (author) - содержит информацию об авторе книги и имеет следующие атрибуты:
  - код (code) - код, позволяющий однозначно определить данного автора, т. к. значения остальных атрибутов (ФИО) могут совпадать;
  - фамилия (surname) - фамилия автора;
  - имя (name) - имя автора;
  - отчество (secname) - отчество автора;
  - инициалы (shortname, shortsecname) - инициалы автора;
- издательство (publ) - содержит информацию об издательстве книги и имеет следующие атрибуты:
  - код (code) - код, позволяющий однозначно определить издательство;
  - имя (name) - название издательства;
  - сокращенное обозначение издательства (shortname);
- книга (book) содержит информацию о книге и имеет следующие атрибуты:
  - код (code) - код, позволяющий однозначно определить книгу;

- имя (name) - название книги;
- количество страниц (pages) - количество страниц книги;
- год издания(year) - год издания книги;
- издательство (publ) - объект класса publ;
- авторы (authors) - список объектов класса author;

При программировании удобно описывать каждый класс в отдельном модуле (файле).

Как видно из приведенного списка, все классы имеют сходные атрибуты - code и name. В этом случае можно выделить их в отдельный класс, от которого будут унаследованы все перечисленные классы. Подобное выделение общих свойств классов в общий класс-предок называется обобщением (generalization). Назовем этот класс «general».

```

1 | #-*- coding:utf-8 -*-
2 | class general:
3 |     def __init__(self,code=0,name=''):
4 |         self.setCode(code)
5 |         self.setName(name)
6 |     def setCode(self,value):self.__code=value
7 |     def setName(self,value):self.__name=value
8 |     def getCode(self):return self.__code
9 |     def getName(self):return self.__name

```

Здесь приведен листинг модуля «general.py». Данный модуль включает в себя описание класса «general».

```

1 | #-*- coding:utf-8 -*-

```

В первой строке задается кодировка файла программы. Это необходимо, если в тексте программы используются символы кириллицы. В данном случае используется кодировка utf-8.

Описание класса:

```

2 | class general:

```

Далее описывается конструктор класса.

```

3 | def __init__(self,code=0,name=''):
4 |     self.setCode(code)
5 |     self.setName(name)

```

В данном случае он позволяет при создании объекта класса задать начальные значения атрибутов класса. В заголовке функции перечислены входные параметры (code,name) и их значения по умолчанию. Первый параметр (self) является ссылкой объекта на самого себя для того, чтобы внутри своих методов обращаться к своим атрибутам и методам, как это сделано в строках 4 и 5. Поля и методы, название которых начинается с двойного подчеркивания являются приватными. Если мы хотим сделать доступными для просмотра и редактирования, нужно описать соответствующие открытые методы. Эти методы принято называть по названию соответствующего атрибута, добавляя вначале слова «get» и «set» для просмотра и редактирования соответственно.

```

6 | def setCode(self,value):self.__code=value
7 | def setName(self,value):self.__name=value

```

Get-методы возвращают в качестве результата значения атрибутов, но может производить и дополнительные действия, например, если значение поля является вычисляемым, зависит от значений других атрибутов, производить соответствующие вычисления и только потом выдавать результат.

```

8 | def getCode(self):return self.__code
9 | def getName(self):return self.__name

```

Также set-методы могут не только присваивать атрибуту значение, заданное пользователем в качестве входного параметра метода, но и производить проверку правильности вводимого значения (например, если значение должно попадать в определенный, диапазон), реагировать на ввод неправильных данных различными способами (сообщение об ошибке, установка значения по умолчанию и т. д.).

В нашем примере значения get и set просто возвращают и устанавливают значения атрибутов.

Далее опишем класс «author» в модуле «author.py»

```

1 | #-*- coding:utf-8 -*-
2 | from general import general
3 |

```

```

4 class author(general):
5     def __init__(self,code=0,surname='',name='',secname='',shortname='',shortsecname=''):
6         general.__init__(self,code,name)
7         self.setSurname(surname)
8         self.setSecname(secname)
9         self.setShortname(shortname)
10        self.setShortsecname(shortsecname)
11
12    def setSurname(self,value):self.__surname=value
13    def setSecname(self,value):self.__secname=value
14    def setShortname(self,value):self.__shortname=value
15    def setShortsecname(self,value):self.__shortsecname=value
16
17    def getSurname(self):return self.__surname
18    def getSecname(self):return self.__secname
19    def getShortname(self):return self.__shortname
20    def getShortsecname(self):return self.__shortsecname
21    def getBibliostr(self):
22        s=self.getSurname()
23        if self.getShortname():s+=' %s.'%(self.getShortname(),)
24        if self.getShortsecname():s+=' %s.'%(self.getShortsecname(),)
25        return s

```

В строке 2 подключается модуль «general.py», содержащий класс «general», от которого унаследован класс «author».

```

2 from general import general

```

Если класс наследуется от другого класса, то название класса предка указывается в скобках после названия наследуемого класса при его описании.

```

3 class author(general):

```

Класс может быть унаследован от нескольких классов (множественное наследование), при этом все классы предки перечисляются в скобках через запятую. Чтобы класс унаследовал все свойства класса предка, в его конструкторе необходимо вызвать конструктор класса предка.

```
5 |     general.__init__(self,code,name)
```

Аналогичным образом описываем класс «publ» в модуле «publ.py»

```
1 | -*- coding:utf-8 -*-
2 | from general import general
3 |
4 | class publ(general):
5 |     def __init__(self,code=0,name='',shortname=''):
6 |         general.__init__(self,code,name)
7 |         self.setShortname(shortname)
8 |     def setShortname(self,value):self.__shortname=value
9 |     def getShortname(self):return self.__shortname
```

При описании класса «book», обратим внимание на то, что книга может иметь несколько авторов, следовательно одним из атрибутов книги станет не объект класса «author», а список таких объектов. Кроме того необходима возможность добавлять нового автора в список и удалять из списка и доступ к атрибутам отдельного автора по его коду. Целесообразно выделить список авторов в отдельный класс. В дальнейшем нам понадобятся аналогичные списки книг и издательств. Таким образом все общие атрибуты и методы этих списков нужно выделить в их общий класс-предок. Назовем его «generalList» и опишем в модуле «generallist.py».

```
1 | -*- coding:utf-8 -*-
2 | class generalList:
3 |     def __init__(self):self.__list=[]
4 |     def clear(self):self.__list=[]
5 |     def findByCode(self,code):
6 |         for l in self.__list:
7 |             if l.getCode()==code:
8 |                 return l
9 |             break
10 |     def getCodes(self):return [s.getCode() for s in self.__list]
11 |     def appendList(self,value):self.__list.append(value)
12 |     def removeList(self,code):
13 |         for s in self.__list:
14 |             if s.getCode()==code:self.__list.remove(s)
15 |     def getName(self,code):return self.findByCode(code).getName()
```

Объекты будут храниться в списке «list», который инициализируется в конструкторе.

```
3 | def __init__(self):self.__list=[]
```

Определим общие для всех необходимых нам списков методы. Метод «clear()» очищает список.

```
4 | def clear(self):self.__list=[]
```

Метод «findByCode(code)» позволяет получить ссылку на объект из списка по значению его атрибута «code», т. к. все наши классы, унаследованные от класса «general» имеют этот атрибут.

```
5 | def findByCode(self,code):
6 |     for l in self.__list:
7 |         if l.getCode()==code:
8 |             return l
9 |         break
```

Метод «getCodes()» возвращает список кодов объектов, находящихся в списке объектов. Это нужно для перебора объектов в цикле.

```
10 | def getCodes(self):return [s.getCode() for s in self.__list]
```

Метод «appendList(value)» добавляет объект в список.

```
16 | def appendList(self,value):self.__list.append(value)
```

Метод «removeList(code)» удаляет из списка объект с заданным значением атрибута «code».

```
17 | def removeList(self,code):
18 |     for s in self.__list:
19 |         if s.getCode()==code:self.__list.remove(s)
```

Метод «getName(code)», как и аналогичный метод класса «general» возвращает значение атрибута «name» объекта с заданным значением атрибута «code». При этом используется ранее описанный метод «findByCode(code)», возвращающий ссылку на объект, затем вызывается метод «getName()» полученного объекта.

```
1 | def getName(self,code):return self.findByCode(code).getName()
```

От класса «generalList» унаследуем класс «authorList», который будет содержать список авторов и все функции для доступа к объектам списка и их атрибутам, в модуле «authorlist.py». Определим в нем только функции, необходимые в книгах, т. е. добавление и удаление автора и просмотр данных об авторе.

```
1 | #-*- coding:utf-8 -*-
2 | from generalist import generalList
3 |
4 | class authorList(generalList):
5 |     def getSurname(self,code):return self.findByCode(code).getSurname()
6 |     def getSecname(self,code):return self.findByCode(code).getSecname()
7 |     def getShortname(self,code):return self.findByCode(code).getShortname()
8 |     def getShortsecname(self,code):return self.findByCode(code).getShortsecname()
9 |     def getBibliostr(self,code):return self.findByCode(code).getBibliostr()
10 |    def getListBibliostr(self):
11 |        s=''
12 |        for code in self.getCodes():
13 |            s+=self.getBibliostr(code)+' , '
14 |        if s:s=s[:-2]
15 |        return s
```

В данном случае конструктор класса не нуждается в описании, т. к. он ни чем не отличается от конструктора класса-предка. Аналогично методу «getName(code)» класса «generalList» опишем методы, возвращающие значения остальных атрибутов авторов.

```
5 | def getSurname(self,code):return self.findByCode(code).getSurname()
6 | def getSecname(self,code):return self.findByCode(code).getSecname()
7 | def getShortname(self,code):return self.findByCode(code).getShortname()
8 | def getShortsecname(self,code):return self.findByCode(code).getShortsecname()
9 | def getBibliostr(self,code):return self.findByCode(code).getBibliostr()
```

Также опишем метод «getListBibliostr()» возвращающий строку, содержащую список авторов в формате библиографической строки - фамилии и инициалы. Авторы в списке разделяются запятыми.

```

10 def getListBibliostr(self):
11     s=''
12     for code in self.getCodes():
13         s+=self.getBibliostr(code)+', '
14     if s:s=s[:-2]
15     return s

```

Далее описываем класс «book» в модуле «book.py»

```

1 #-*- coding:utf-8 -*-
2 from general import general
3 from publ import publ
4 from authorlist import authorList
5 class book(general):
6     def __init__(self,code=0,name='',img='',publ=None,year=0,pages=0):
7         general.__init__(self,code,name)
8         self.__authors=authorList()
9         self.setImg(img)
10        self.setPubl(publ)
11        self.setPages(pages)
12        self.setYear(year)
13    def setImg(self,value):self.__img=value
14    def setPubl(self,value):self.__publ=value
15    def setPages(self,value):self.__pages=value
16    def setYear(self,value):self.__year=value
17    def getImg(self):return self.__img
18    def getPublCode(self):
19        if self.__publ:return self.__publ.getCode()
20    def getPublName(self):
21        if self.__publ:return self.__publ.getName()
22        else:return ''
23    def getPublShortname(self):
24        if self.__publ:return self.__publ.getShortname()
25        else:return ''
26    def getPages(self):return self.__pages
27    def getYear(self):return self.__year

```



```

28 def appendAuthor(self,value):self.__authors.appendList(value)
29 def removeAuthor(self,code):self.__authors.removeList(code)
30 def clearAuthors(self):self.__authors.clear()
31 def getAuthorCodes(self):return self.__authors.getCodes()
32 def getAuthorName(self,code):return self.__authors.findByCode(code).getSurname()
33 def getAuthorSurname(self,code):return self.__authors.findByCode(code).getSurname()
34 def getAuthorSecname(self,code):return self.__authors.findByCode(code).getSecname()
35 def getAuthorShortname(self,code):return self.__authors.findByCode(code).getShortname()
36 def getAuthorShortsecname(self,code):return self.__authors.findByCode(code).getShortsecname()
37 def getAuthorsBibliostr(self):return self.__authors.getListBibliostr()
38 def getBibliostr(self):
39     s=self.getAuthorsBibliostr()
40     if self.__publ:spubl=self.getPublShortname()
41     else:spubl=''
42     s+=' %s - %s, %s. - %s c.'%(self.getName(),spubl,str(self.getYear()),str(self.getPages()))
43     return s

```

Импортируем необходимые классы.

```

2 from general import general
3 from publ import publ
4 from authorlist import authorList

```

Объявляем класс «book», унаследовав его от класса «general».

```

6 class book(general):

```

Реализуем конструктор в котором установим начальные значения атрибутов, при этом используется также конструктор класса-предка. Кроме того, в качестве атрибута «authors» (списка авторов книги) класса «book», будет использован объект класса «authorList». объект класса

```

7 def __init__(self,code=0,name='',publ=None,year=0,pages=0):
8     general.__init__(self,code,name)
9     self.__authors=authorList()
10    self.setPubl(publ)
11    self.setPages(pages)
12    self.setYear(year)

```

Значение по умолчанию для атрибута, содержащего ссылку на объект лучше задать «None» - пустое значение.

```
1 | def __init__(self,code=0,name='',publ=None,year=0,pages=0):
```

Определяем методы доступа к собственным атрибутам класса «book».

```
13 | def setPubl(self,value):self.__publ=value
14 | def setPages(self,value):self.__pages=value
15 | def setYear(self,value):self.__year=value
```

```
20 | def getPages(self):return self.__pages
21 | def getYear(self):return self.__year
```

Соблюдение правила инкапсуляции требует скрывать структуру класса и предоставлять доступ к атрибутам включаемых объектов через специально определенные методы. Определим методы доступа к атрибутам класса «book», являющимся объектами - «publ» и «authors». При этом издательства и авторы являются самостоятельными сущностями, поэтому книга не имеет методов редактирования их атрибутов, только методы получения их значений.

```
17 | def getPublCode(self):
18 |     if self.__publ:return self.__publ.getCode()
19 | def getPublName(self):
20 |     if self.__publ:return self.__publ.getName()
21 |     else:return ''
22 | def getPublShortname(self):
23 |     if self.__publ:return self.__publ.getShortname()
24 |     else:return ''
```

```
23 | def appendAuthor(self,value):self.__authors.appendList(value)
24 | def removeAuthor(self,code):self.__authors.removeList(code)
25 | def clearAuthors(self):self.__authors.clear()
26 | def getAuthorCodes(self):return self.__authors.getCodes()
27 | def getAuthorName(self,code):return self.__authors.findByCode(code).getSurname()
28 | def getAuthorSurname(self,code):return self.__authors.findByCode(code).getSurname()
29 | def getAuthorSecname(self,code):return self.__authors.findByCode(code).getSecname()
30 | def getAuthorShortname(self,code):return self.__authors.findByCode(code).getShortname()
31 | def getAuthorShortsecname(self,code):return self.__authors.findByCode(code).getShortsecname()
32 | def getAuthorsBibliostr(self):return self.__authors.getListBibliostr()
```

Опишем метод «getBibliostr()» возвращающий строку, содержащую информацию о книге в формате библиографической строки - сначала список авторов с фамилиями и инициалами через запятую (уже реализовано в классе «authorList»), затем название книги, сокращенное название издательства, год издания, количество страниц.

```
32 def getBibliostr(self):
33     s=self.getAuthorsBibliostr()
34     if self.__publ:spubl=self.getPublShortname()
35     else:spubl=''
36     s+=' %s - %s, %s. - %s c.'%(self.getName(),spubl,str(self.getYear()),str(self.getPages()))
37     return s
```

Теперь напишем программу в файле «main1.py», которая будет создавать объекты от ранее описанных классов и для демонстрации работы будет выводить на экран информацию о книге в виде строки в библиографическом списке.

```
1 #-*- coding:utf-8 -*-
2 from book import book
3 from publ import publ
4 from author import author
5 b1=book(1,'Матричный анализ','matrix.jpeg',year=1989,pages=655)
6 b1.appendauthor(author(1,'Хорн','Роджер',shortname='Р'))
7 b1.appendauthor(author(2,'Джонсон','Чарльз',shortname='Ч'))
8 b1.setpubl(publ(1,'Москва "Мир"', 'М.: Мир'))
9 print b1.getBibliostr()
```

Импортируем необходимые классы.

```
2 from book import book
3 from publ import publ
4 from author import author
```

Создаем объект класса «book» (книга), при этом в переменную «b1» записывается ссылка на объект. При этом авторы и издательство пока не задаются. Эти объекты необходимо сначала создать.

```
5 b1=book(1,'Матричный анализ',year=1989,pages=655)
```

Создаются объекты класса «author» (авторы) и сразу добавляются в список авторов книги.

```
6 | b1.appendAuthor(author(1,'Хорн','Роджер',shortname='Р'))  
7 | b1.appendAuthor(author(2,'Джонсон','Чарльз',shortname='Ч'))
```

Создается объект класса «publ» (издательство) и сразу устанавливается соответствующий атрибут книги.

```
8 | b1.setpubl(publ(1,'Москва "Мир"', 'М.: Мир'))
```

Генерируется и выводится на экран библиографическая строка для данной книги.

```
9 | print b1.getbibliostr()
```

Результатом выполнения программы будет вывод на экран следующей строки:

```
| Хорн Р. Джонсон Ч. Матричный анализ - М.: Мир, 1989. - 655 с.
```

Диаграмма классов данной программы приведена на рисунке (3).

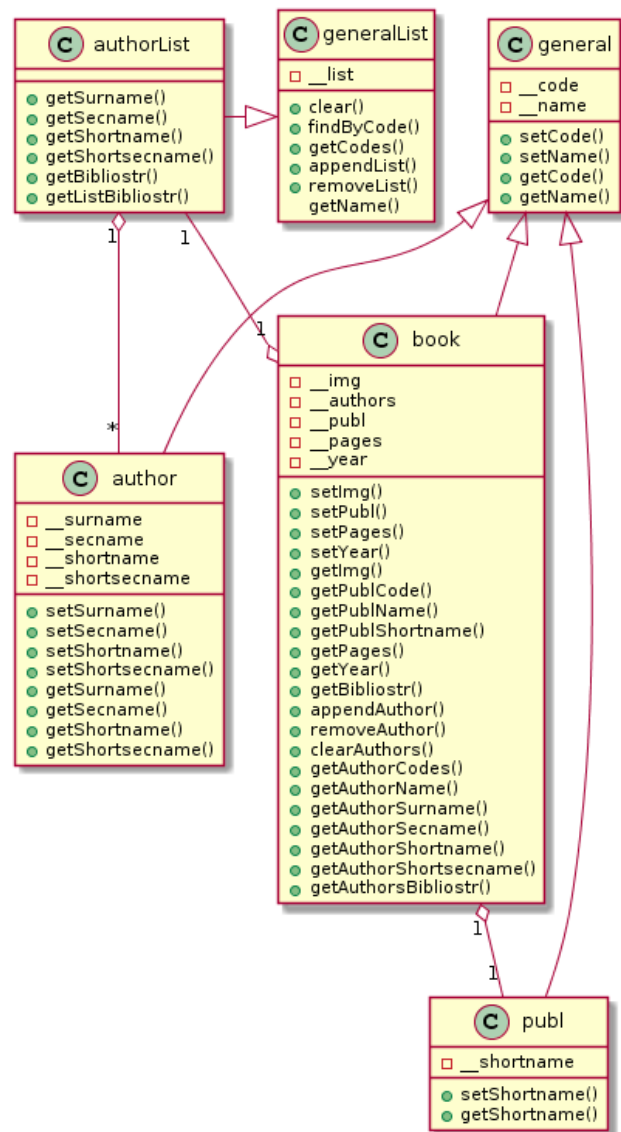


Рис. 3: Диаграмма классов для задания 1

## 4 Задание 2

Написать функции для считывания и записи данных о выбранных сущностях из файла в формате XML, используя при этом классы, полученные при выполнении задания 1. Для полученной программы построить диаграмму классов UML.

Реализовать запросы, рекомендованные преподавателем. Для демонстрации работы функции организовать вывод данных на экран в консоли.

### Пример выполнения

В текстовом редакторе создадим файл, назовем его «old.xml». Запишем в него следующие данные в формате XML:

```
1 <library>
2   <author code="1" surname="Хорн" name="Роджер" shortname="Р"/>
3   <author code="2" surname="Джонсон" name="Чарльз" shortname="Ч"/>
4   <author code="3" surname="Сузи" name="Роман" shortname="Р" shortsecname="А"/>
5   <author code="4" surname="Мякишев" name="Геннадий" secname="Яковлевич" shortname="Г" shortsecname="Я"/>
6   <author code="5" surname="Буховцев" name="Борис" secname="Борисович" shortname="Б" shortsecname="Б"/>
7
8   <publ code="1" name='Москва "Мир"' shortname="М.: Мир"/>
9   <publ code="2" name='Санкт-Петербург "БХВ-Петербург"' shortname="СПб.: БХВ-Петербург"/>
10  <publ code="3" name='Москва "Просвещение"' shortname="М.: Просвещение"/>
11
12  <book code="1" name='Матричный анализ' img='matrix.jpeg' year='1989' pages='655' publ='1'>
13    <author code='1'/>
14    <author code='2'/>
15  </book>
16
17  <book code="2" name='Python' img='cuzypython.jpeg' year='2002' pages='768' publ='2'>
18    <author code='3'/>
19  </book>
20
21  <book code="3" name='Физика: Учебник для 11 класса средней школы' img='fizika.jpeg' year='1991' pages='254' publ='3'>
22    <author code='4'/>
23    <author code='5'/>
```

```
24 | </book>
25 | </library>
```

Для хранения в программе считанных данных из XML-файла нам потребуется несколько объектов классов «author», «publ», «book», которые удобно хранить в списках, как это было со списком авторов одной книги. Список таких объектов должен иметь функции

- создания нового объекта в списке, для этого нужна также функция, определяющая код создаваемого объекта;
- добавления существующего объекта в список;
- возможность просмотра значений атрибутов объекта;
- возможность редактирования значений атрибутов объекта;
- удаление объекта из списка.

Назовем классы реализующие эти списки, соответственно, «authorListEdit», «publListEdit», «bookListEdit». Для начала определим их общий класс-предок «generalListEdit» в модуле «generallistedit.py»

```
1  # -*- coding:utf-8 -*-
2  from generallist import generalList
3
4  class generalListEdit(generalList):
5      def getNewCode(self):
6          m=0
7          for c in self.getCodes():
8              if c>m:m=c
9          return m+1
10 def setName(self,code,value):return self.findByCode(code).setName(value)
```

Метод «getNewCode()» позволяет получить вычислить следующий код при создании нового объекта. В данном случае ищется максимальный код из списка кодов всех объектов, и в качестве результата возвращается его значение, увеличенное на 1. Например, если в списке присутствуют авторы с кодами 1, 2, 3, следующий автор будет создан с кодом 4. Коды могут идти не по порядку и не подряд, это связано с возможным удалением объектов и различным порядком считывания данных из внешних файлов.

```

5 | def getNewCode(self):
6 |     m=0
7 |     for c in self.getCodes():
8 |         if c>m:m=c
9 |     return m+1

```

Для класса «author» уже есть класс «authorlist», реализующий возможности

- добавления существующего объекта в список;
- возможность просмотра значений атрибутов объекта;
- удаление объекта из списка.

Таким образом, «authorListEdit» можно унаследовать от «authorList», добавив недостающие функции. Опишем его в модуле «authorlistedit.py».

```

1 | #-*- coding:utf-8 -*-
2 | from authorlist import authorList
3 | from generallistedit import generalListEdit
4 | from author import author
5 | class authorListEdit(authorList,generalListEdit):
6 |     def newRec(self,code=0,surname='',name='',secname='',shortname='',shortsecname=''):self.appendList(author(code,surname,name,secnam
7 |     def setSurname(self,code,value):self.findByCode(code).setSurname(value)
8 |     def setName(self,code,value):self.findByCode(code).setName(value)
9 |     def setSecname(self,code,value):self.findByCode(code).setSecname(value)
10 |    def setShortname(self,code,value):self.findByCode(code).setShortname(value)
11 |    def setShortsecname(self,code,value):self.findByCode(code).setShortsecname(value)

```

Классы «publListEdit», «bookListEdit» создадим с нуля в соответствующих модулях.

Модуль «publitedit.py»:

```

1 | #-*- coding:utf-8 -*-
2 | from publ import publ
3 | from generallistedit import generalListEdit

```



```

4
5 class publListEdit(generallistEdit):
6     def newRec(self,code=0,name='',shortname=''):self.appendList(publ(code,name,shortname))
7     def getShortname(self,code):return self.findByCode(code).getShortname()
8     def setShortname(self,code,value):self.findByCode(code).setShortname(value)

```

Модуль «booklistedit.py»:

```

1 #-*- coding:utf-8 -*-
2 from book import book
3 from generallistedit import generallistEdit
4
5 class bookListEdit(generallistEdit):
6     def newRec(self,code=0,name='',img='',publ=None,year=0,pages=0):self.appendList(book(code,name,img,publ,year,pages))
7     def setImg(self,code,value):self.findByCode(code).setImg(value)
8     def setPubl(self,code,value):self.findByCode(code).setPubl(value)
9     def setPages(self,code,value):self.findByCode(code).setPages(value)
10    def setYear(self,code,value):self.findByCode(code).setYear(value)
11    def appendAuthor(self,code,value):self.findByCode(code).appendAuthor(value)
12    def removeAuthor(self,bcode,acode):self.findByCode(bcode).removeAuthor(acode)
13    def clearAuthors(self,code):self.findByCode(code).clearAuthors()
14    def getImg(self,code):return self.findByCode(code).getImg()
15    def getPublCode(self,code):return self.findByCode(code).getPublCode()
16    def getPublName(self,code):return self.findByCode(code).getPublName()
17    def getPublShortname(self,code):return self.findByCode(code).getPublName()
18    def getPages(self,code):return self.findByCode(code).getPages()
19    def getYear(self,code):return self.findByCode(code).getYear()
20    def getAuthorSurname(self,bcode,acode):return self.findByCode(bcode).getAuthorSurname(acode)
21    def getAuthorName(self,bcode,acode):return self.findByCode(bcode).getAuthorName(acode)
22    def getAuthorSecname(self,bcode,acode):return self.findByCode(bcode).getAuthorSecname(acode)
23    def getAuthorShortname(self,bcode,acode):return self.findByCode(bcode).getAuthorShortname(acode)
24    def getAuthorShortsecname(self,bcode,acode):return self.findByCode(bcode).getAuthorShortsecname(acode)
25    def getAuthorCodes(self,code):return self.findByCode(code).getAuthorCodes()
26    def getAuthorsBibliostr(self,code):return self.findByCode(code).getAuthorsBibliostr()
27    def getBibliostr(self,code):return self.findByCode(code).getBibliostr()

```

Все эти списки объектов будут храниться в объекте класса «library» в модуле «library.py». Данный класс также содержит функции уничтожения объектов.

```
1 #-*- coding:utf-8 -*-
2 from authorlistedit import authorListEdit
3 from publlistedit import publListEdit
4 from booklistedit import bookListEdit
5
6 class library:
7     def __init__(self):
8         self.__authors=authorListEdit()
9         self.__publs=publListEdit()
10        self.__books=bookListEdit()
11    def removeAuthor(self,code):
12        self.__authors.removeList(code)
13        for c in self.__books.getCodes():
14            self.__books.removeAuthor(c,code)
15    def removePubl(self,code):
16        self.__publs.removeList(code)
17        for c in self.__books.getCodes():
18            if self.getBookPublCode(c)==code:
19                self.__books.setPubl(c,None)
20    def clear(self):
21        self.__books.clear()
22        self.__authors.clear()
23        self.__publs.clear()
24    def newAuthor(self,code=0,surname='',name='',secname='',shortname='',shortsecname=''):self.__authors.newRec(code,surname,name,secname,shortname,shortsecname)
25    def findAuthorByCode(self,code):return self.__authors.findByCode(code)
26    def getAuthorNewCode(self):return self.__authors.getNewCode()
27    def getAuthorCodes(self):return self.__authors.getCodes()
28    def getAuthorName(self,code):return self.__authors.getName(code)
29    def getAuthorSurname(self,code):return self.__authors.getSurname(code)
30    def getAuthorSecname(self,code):return self.__authors.getSecname(code)
31    def getAuthorShortname(self,code):return self.__authors.getShortname(code)
32    def getAuthorShortsecname(self,code):return self.__authors.getShortsecname(code)
33    def getAuthorBibliostr(self,code):return self.__authors.getBibliostr(code)
```

```

34 def getAuthorNewCode(self):return self.__authors.getNewCode()
35 def setAuthorName(self,code,value):self.__authors.setName(code,value)
36 def setAuthorSurname(self,code,value):self.__authors.setSurname(code,value)
37 def setAuthorSecname(self,code,value):self.__authors.setSecname(code,value)
38 def setAuthorShortname(self,code,value):self.__authors.setShortname(code,value)
39 def setAuthorShortsecname(self,code,value):self.__authors.setShortsecname(code,value)
40
41 def newPubl(self,code=0,name='',shortname=''):self.__publs.newRec(code,name,shortname)
42 def findPublByCode(self,code):return self.__publs.findByCode(code)
43 def getPublNewCode(self):return self.__publs.getNewCode()
44 def getPublCodes(self):return self.__publs.getCodes()
45 def getPublName(self,code):return self.__publs.getName(code)
46 def getPublShortname(self,code):return self.__publs.getShortname(code)
47 def getPublNewCode(self):return self.__publs.getNewCode()
48 def setPublName(self,code,value):self.__publs.setName(code,value)
49 def setPublShortname(self,code,value):self.__publs.setShortname(code,value)
50
51 def removeBook(self,code):self.__books.removeList(code)
52 def newBook(self,code=0,name='',img='',publ=None,year=0,pages=0):self.__books.newRec(code,name,img,publ,year,pages)
53 def findBookByCode(self,code):return self.__books.findByCode(code)
54 def appendBookAuthor(self,bcode,value):self.__books.appendAuthor(bcode,value)
55 def removeBookAuthor(self,bcode,acode): self.__books.removeAuthor(bcode,acode)
56 def clearBookAuthors(self,bcode):self.__books.clearAuthors(bcode)
57 def setBookName(self,code,value):self.__books.setName(code,value)
58 def setBookImg(self,code,value):self.__books.setImg(code,value)
59 def setBookPubl(self,code,pcode):self.__books.setPubl(code,self.findPublByCode(pcode))
60 def setBookPages(self,code,value):self.__books.setPages(code,value)
61 def setBookYear(self,code,value):self.__books.setYear(code,value)
62 def getBookCodes(self):return self.__books.getCodes()
63 def getBookNewCode(self):return self.__books.getNewCode()
64 def getBookName(self,code):return self.__books.getName(code)
65 def getBookImg(self,code):return self.__books.getImg(code)
66 def getBookPublCode(self,code):return self.__books.getPublCode(code)
67 def getBookPublName(self,code):return self.__books.getPublName(code)
68 def getBookPublShortname(self,code):return self.__books.getPublName(code)
69 def getBookPages(self,code):return self.__books.getPages(code)

```

```

70 def getBookYear(self,code):return self.__books.getYear(code)
71 def getBookAuthorSurname(self,bcode,acode):return self.__books.getAuthorSurname(bcode,acode)
72 def getBookAuthorName(self,bcode,acode):return self.__books.getAuthorName(bcode,acode)
73 def getBookAuthorSecname(self,bcode,acode):return self.__books.getAuthorSecname(bcode,acode)
74 def getBookAuthorShortname(self,bcode,acode):return self.__books.getAuthorShortname(bcode,acode)
75 def getBookAuthorShortsecname(self,bcode,acode):return self.__books.getAuthorShortsecname(bcode,acode)
76 def getBookAuthorCodes(self,code):return self.__books.getAuthorCodes(code)
77 def getBookNewCode(self):return self.__books.getNewCode()
78 def getBookAuthorsBibliostr(self,code):return self.__books.getAuthorsBibliostr(code)
79 def getBookBibliostr(self,code):return self.__books.getBibliostr(code)

```

Теперь реализуем чтение данных из XML-файла. Для этого опишем класс «dataxml» в модуле «dataxml.py», содержащий функции чтения и записи данных в формате XML.

Модуль «dataxml.py»:

```

1  -*- coding:utf-8 -*-
2  import os,xml.dom.minidom
3  class dataxml:
4      def read(self,inp,lib):
5          dom=xml.dom.minidom.parse(inp)
6          dom.normalize()
7          for node in dom.childNodes[0].childNodes:
8              if (node.nodeType==node.ELEMENT_NODE)and(node.nodeName=='author'):
9                  code,surname,name,secname,shortname,shortsecname=0,"","","","",""
10                 for t in node.attributes.items():
11                     if t[0]=="code":code=int(t[1])
12                     if t[0]=="name":name=t[1]
13                     if t[0]=="surname":surname=t[1]
14                     if t[0]=="secname":secname=t[1]
15                     if t[0]=="shortname":shortname=t[1]
16                     if t[0]=="shortsecname":shortsecname=t[1]
17                 lib.newAuthor(code,surname,name,secname,shortname,shortsecname)
18             if (node.nodeType==node.ELEMENT_NODE)and(node.nodeName=='publ'):
19                 code,name,shortname=0,"",""
20                 for t in node.attributes.items():
21                     if t[0]=="code":code=int(t[1])

```

```

22         if t[0]=="name":name=t[1]
23         if t[0]=="shortname":shortname=t[1]
24     lib.newPubl(code,name,shortname)
25 if (node.nodeType==node.ELEMENT_NODE)and(node.nodeName=='book'):
26     code,name,img,publ,year,pages=0,'','','',None,0,0
27     for t in node.attributes.items():
28         if t[0]=="code":code=int(t[1])
29         if t[0]=="name":name=t[1]
30         if t[0]=="img":img=t[1]
31         if t[0]=="year":year=int(t[1])
32         if t[0]=="pages":pages=int(t[1])
33         if t[0]=="publ":publ=lib.findPublByCode(int(t[1]))
34     lib.newBook(code,name,img,publ,year,pages)
35     for n in node.childNodes:
36         if (n.nodeType==n.ELEMENT_NODE)and(n.nodeName=='author'):
37             for t in n.attributes.items():
38                 if t[0]=="code":author=lib.findAuthorByCode(int(t[1]))
39                 lib.appendBookAuthor(code,author)
40 def write(self,out,lib):
41     dom=xml.dom.minidom.Document()
42     root=dom.createElement("library")
43     dom.appendChild(root)
44     for c in lib.getAuthorCodes():
45         aut=dom.createElement("author")
46         aut.setAttribute('code',str(c))
47         aut.setAttribute('surname',lib.getAuthorSurname(c))
48         aut.setAttribute('name',lib.getAuthorName(c))
49         aut.setAttribute('secname',lib.getAuthorSecname(c))
50         aut.setAttribute('shortname',lib.getAuthorShortname(c))
51         aut.setAttribute('shortsecname',lib.getAuthorShortsecname(c))
52     root.appendChild(aut)
53     for c in lib.getPublCodes():
54         pub=dom.createElement("publ")
55         pub.setAttribute('code',str(c))
56         pub.setAttribute('name',lib.getPublName(c))
57         pub.setAttribute('shortname',lib.getPublShortname(c))

```

```

58     root.appendChild(pub)
59 for c in lib.getBookCodes():
60     bk=dom.createElement("book")
61     bk.setAttribute('code',str(c))
62     bk.setAttribute('name',lib.getBookName(c))
63     bk.setAttribute('img',lib.getBookImg(c))
64     bk.setAttribute('year',str(lib.getBookYear(c)))
65     bk.setAttribute('pages',str(lib.getBookPages(c)))
66     bk.setAttribute('publ',str(lib.getBookPublCode(c)))
67     for ac in lib.getBookAuthorCodes(c):
68         aut=dom.createElement("author")
69         aut.setAttribute('code',str(ac))
70         bk.appendChild(aut)
71     root.appendChild(bk)
72 f = open(out,"w")
73 f.write(dom.toprettyxml())

```

Данный пример для Python3, при работе в Python2.7 в последней строке нужно указывать кодировку

```

1 f.write(dom.toprettyxml(encoding='utf-8'))

```

Теперь напишем программу в файле «main2.py», которая будет создавать объекты классов «library» и «dataxml»

```

1 from library import library
2 from dataxml import dataxml as data
3
4 lib1=library()
5 dat1=data()
6 dat1.read('old.xml',lib1)
7 dat1.write('new.xml',lib1)
8 for s in lib1.getBibliostrs():
9     print s

```

Результат работы программы:

- 1 Хорн Р. Джонсон Ч. Матричный анализ - М.: Мир, 1989. - 655 с.  
2 Сузи Р. А. Python - СПб.: БХВ-Петербург, 2002. - 768 с.  
3 Мякишев Г. Я. Буховцев Б. Б. Физика: Учебник для 11 класса средней школы - М.: Просвещение, 1991. - 254 с.

На диске также появится файл «new.xml» содержащий следующее:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <library>
3     <author code="1" name="Роджер" secname="" shortname="P" shortsecname="" surname="Хорн"/>
4     <author code="2" name="Чарльз" secname="" shortname="Ч" shortsecname="" surname="Джонсон"/>
5     <author code="3" name="Роман" secname="" shortname="P" shortsecname="A" surname="Сузи"/>
6     <author code="4" name="Геннадий" secname="Яковлевич" shortname="Г" shortsecname="Я" surname="Мякишев"/>
7     <author code="5" name="Борис" secname="Борисович" shortname="Б" shortsecname="Б" surname="Буховцев"/>
8     <publ code="1" name="Москва &quot;Мир&quot;" shortname="М.: Мир"/>
9     <publ code="2" name="Санкт-Петербург &quot;БХВ-Петербург&quot;" shortname="СПб.: БХВ-Петербург"/>
10    <publ code="3" name="Москва &quot;Просвещение&quot;" shortname="М.: Просвещение"/>
11    <book code="1" img="matrix.jpeg" name="Матричный анализ" pages="655" publ="1" year="1989">
12        <author code="1"/>
13        <author code="2"/>
14    </book>
15    <book code="2" img="cuzypython.jpeg" name="Python" pages="768" publ="2" year="2002">
16        <author code="3"/>
17    </book>
18    <book code="3" img="fizika.jpeg" name="Физика: Учебник для 11 класса средней школы" pages="254" publ="3" year="1991">
19        <author code="4"/>
20        <author code="5"/>
21    </book>
22 </library>
```

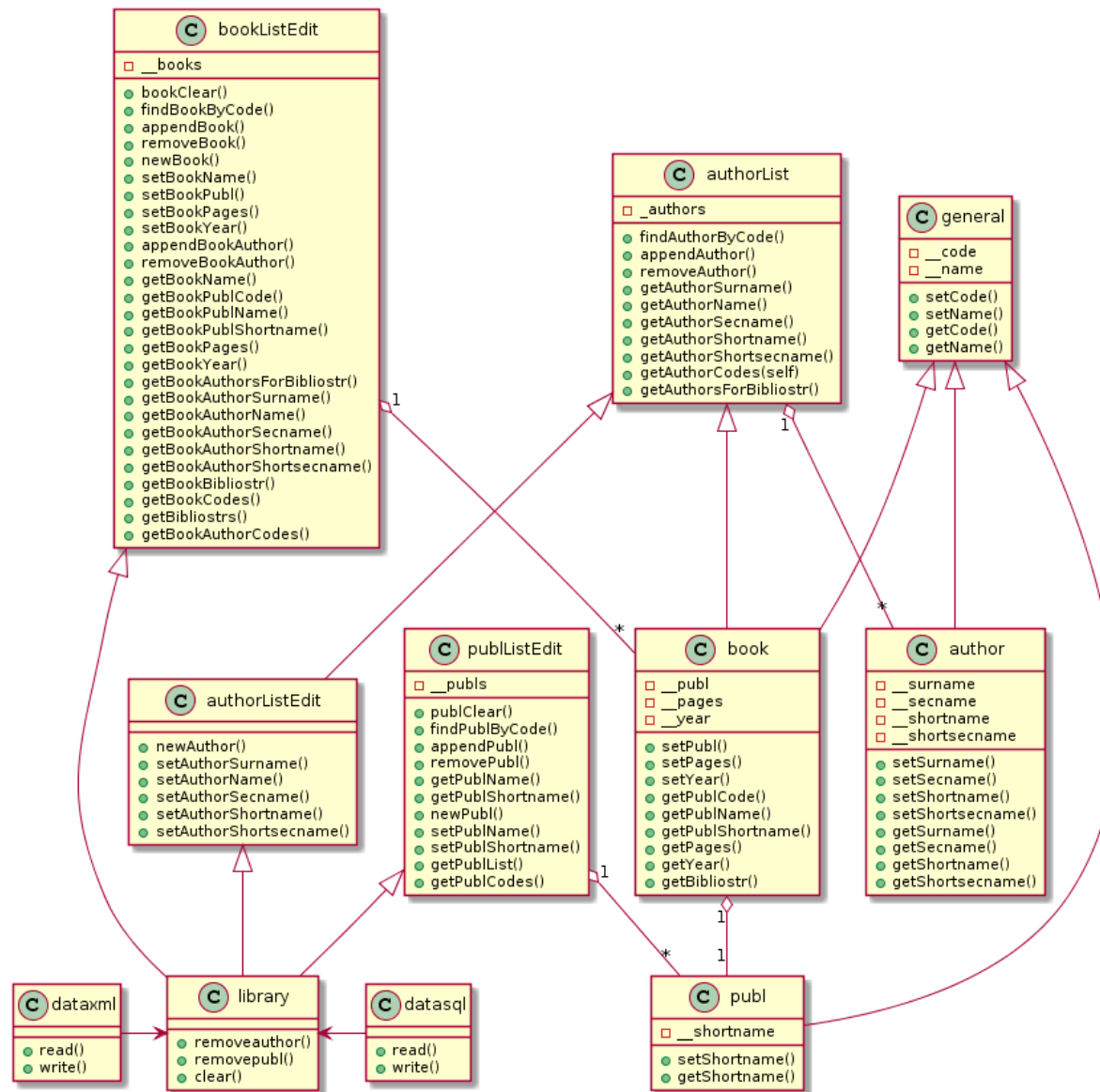


Рис. 4: Диаграмма классов для задания 2 и 3



## 5 Задание 3

Написать функции для считывания и записи данных о выбранных сущностях из базы данных SQLite, используя при этом классы, полученные при выполнении заданиях 1 и 2. Для полученной программы построить диаграмму классов UML.

Реализовать запросы, рекомендованные преподавателем. Для демонстрации работы функции организовать вывод данных на экран в консоли.

### Пример выполнения

Теперь реализуем чтение данных из базы данных SQLite. Для этого опишем класс «datasql» в модуле «datasql.py», содержащий функции чтения и записи данных в формате SQLite.

Модуль «datasql.py»:

```
1  -*- coding:utf-8 -*-
2  import os
3  import sqlite3 as db
4
5  emptydb = ""
6  PRAGMA foreign_keys = ON;
7
8  create table author
9  (code integer primary key,
10   surname text,
11   name text,
12   secname text,
13   shortname text,
14   shortsecname text);
15
16  create table publ
17  (code integer primary key,
18   name text,
19   shortname text);
20
```

```

21 create table book
22 (code integer primary key,
23  name text,
24  img text,
25  year integer,
26  pages integer,
27  publ integer references publ(code) on update cascade on delete set null);
28
29 create table book_author
30 (code integer primary key autoincrement,
31  book integer references book(code) on update cascade on delete cascade,
32  author integer references author(code) on update cascade on delete cascade,
33  unique(book,author));
34 """
35
36 class datasql:
37     def read(self,inp,lib):
38         conn = db.connect(inp)
39         curs = conn.cursor()
40         curs.execute('select code,surname,name,secname,shortname,shortsecname from author')
41         data=curs.fetchall()
42         for r in data:lib.newAuthor(r[0],r[1],r[2],r[3],r[4],r[5])
43         curs.execute('select code,name,shortname from publ')
44         data=curs.fetchall()
45         for r in data:lib.newPubl(r[0],r[1],r[2])
46         curs.execute('select code,name,img,year,pages,publ from book')
47         data=curs.fetchall()
48         for r in data:lib.newBook(r[0],r[1],r[2],lib.findPublByCode(int(r[5])),r[3],r[4])
49         curs.execute('select book,author from book_author')
50         data=curs.fetchall()
51         for r in data:lib.appendBookAuthor(r[0],lib.findAuthorByCode(r[1]))
52         conn.close()
53     def write(self,out,lib):
54         conn = db.connect(out)
55         curs = conn.cursor()
56         curs.executescript(emptydb)

```

```

57 for c in lib.getAuthorCodes():
58     curs.execute("insert into author(code,surname,name,secname,shortname,shortsecname) values('%s','%s','%s','%s','%s','%s')"%(
59         str(c),
60         lib.getAuthorSurname(c),
61         lib.getAuthorName(c),
62         lib.getAuthorSecname(c),
63         lib.getAuthorShortname(c),
64         lib.getAuthorShortsecname(c)))
65 for c in lib.getPublCodes():
66     curs.execute("insert into publ(code,name,shortname) values('%s','%s','%s')"%(
67         str(c),lib.getPublName(c),lib.getPublShortname(c)))
68 for c in lib.getBookCodes():
69     curs.execute("insert into book(code,name,img,year,pages,publ) values('%s','%s','%s','%s','%s','%s')"%(
70         str(c),
71         lib.getBookName(c),
72         lib.getBookImg(c),
73         str(lib.getBookYear(c)),
74         str(lib.getBookPages(c)),
75         str(lib.getBookPublCode(c))))
76 for ac in lib.getBookAuthorCodes(c):
77     curs.execute("insert into book_author(book,author) values('%s','%s')"%(str(c),str(ac)))
78 conn.commit()
79 conn.close()

```

Теперь напишем программу в файле «main2.py», которая будет создавать объекты классов «library» и «datasql»

```

1 from library import library
2 from datasql import datasql as data
3
4 lib1=library()
5 dat1=data()
6 dat1.read('old.sqlite',lib1)
7 dat1.write('new.sqlite',lib1)
8 for s in lib1.getBibliostrs():
9     print s

```

Результат работы программы:

- <sup>1</sup> | Хорн Р. Джонсон Ч. Матричный анализ – М.: Мир, 1989. – 655 с.
- <sup>2</sup> | Сузи Р. А. Python – СПб.: БХВ-Петербург, 2002. – 768 с.
- <sup>3</sup> | Мякишев Г. Я. Буховцев Б. Б. Физика: Учебник для 11 класса средней школы – М.: Просвещение, 1991. – 254 с.

## 6 Задание 4

С использованием библиотеки для программирования графического интерфейса QT сделать форму для вывода данных, в табличной форме, хранящихся в формате XML и в базе данных SQLite, с использованием функций из заданий 2 и 3.

### Пример выполнения

Сделаем форму примерно такого вида, как показано на рисунке (5).

Библиотека

Еле

книги авторы издательства

	название	обложка	авторы	издательство	год	страницы
1	Матричный анализ	matrix.jpeg	Хорн Р., Джонсон Ч.	Москва "Мир"	1989	655
2	Python	cuzypython.jpeg	Сузи Р. А.	Санкт-Петербург "БХВ-Петербург"	2002	768
3	Физика: Учебник для 11 класса средней школы	fizika.jpeg	Мякишев Г. Я., Буховцев Б. Б.	Москва "Просвещение"	1991	254

название Python

обложка cuzypython.jpeg

авторы Сузи Р. А.

Хорн Р.

издательство Санкт-Петербург "БХВ-Петербург"

год 2002

страницы 768

Добавить

найти

Изменить

Удалить

удалить

добавить

Рис. 5: Форма для редактирования данных

Опишем виджеты для отображения данных. Все они должны иметь возможность устанавливать соответствие между строками данных и кодами записей, отображаемых в них. Для этого опишем класс «rowCode» в модуле «rowcode.py» Список упорядоченных пар (tuple) «list», хранит соответствия между номерами строк таблицы и кодами отображаемых в них записей.

```

1 class rowCode:
2     def __init__(self):self.__list=[]
3     def clear(self):self.__list=[]
4     def appendRowCode(self,row,code):self.__list.append([row,code])
5     def updateRow(self,row):
6         for t in self.__list:
7             if t[0]>row:t[0]-=1

```

```

8 def removeRow(self,row):
9     for t in self.__list:
10         if t[0]==row:self.__list.remove(t)
11     self.updateRow(row)
12 def removeCode(self,code):
13     for t in self.__list:
14         if t[1]==code:self.__list.remove(t)
15 def getCode(self,row):
16     for t in self.__list:
17         if t[0]==row:return t[1]
18 def getCodes(self):return [t[1] for t in self.__list]
19 def getRow(self,code):
20     for t in self.__list:
21         if t[1]==code:return t[0]

```

Большенство виджетов содержат в себе ссылку на объект класса «library». Опишем модуль, «libwidget.py».

```

1 class libWidget:
2     def __init__(self,library=None):
3         self.__library=library
4     def getLibrary(self):return self.__library

```

Определим классы для отображения данных в табличной форме. Для каждой сущности определим отдельный класс. У всех этих классов есть общие атрибуты и методы, которые опишем в общем классе предке «dbTableWidget» в модуле «dbtablewidget.py».

```

1 from PyQt5.QtWidgets import QTableWidgetItem
2 from rowcode import rowCode
3 from libwidget import libWidget
4
5 class dbTableWidget(QTableWidgetItem,libWidget):
6     def __init__(self,library,parent=None,header=[]):
7         QTableWidgetItem.__init__(self)
8         libWidget.__init__(self,library)
9         self.__rowCode=rowCode()

```

```

10     self.setHeader(header)
11     self.update()
12 def setHeader(self, value):
13     self.setColumnCount(len(value))
14     self.setHorizontalHeaderLabels(value)
15 def clearContents(self):
16     self.__rowCode.clear()
17     QTableWidgetItem.clearContents(self)
18 def getCodes(self): return self.__rowCode.getCodes()
19 def getCurrentCode(self): return self.__rowCode.getCode(self.currentRow())
20 def appendRowCode(self, row, code): self.__rowCode.appendRowCode(row, code)
21 def update(self): pass

```

Данный класс наследуется от стандартного класса библиотеки QT «QTableWidgetItem». Подключаем модули библиотеки PyQt5:

```

1 from PyQt5.QtWidgets import QTableWidgetItem

```

Объявляем класс «dbTableWidget», наследуемый от «QTableWidgetItem» и описываем конструктор нового класса. При создании нового объекта ему передается ссылка на объект «library», содержащий отображаемые данные. Класс «library» был разработан при выполнении задания 2. Через параметр «parent» передается ссылка на виджет-хозяин пекущего виджета, например, если таблица расположена в диалоговом окне. В строке 6 вызывается конструктор класса-предка.

Опишем метод «setHeader», в котором установим количество колонок таблицы функцией «setColumnCount» класса «QTableWidgetItem» и заголовки колонок таблицы функцией «setHorizontalHeaderLabels» того же класса.

```

11 def setHeader(self, value):
12     self.setColumnCount(len(value))
13     self.setHorizontalHeaderLabels(value)

```

Переопределим функцию «clearContents» класса «QTableWidgetItem», которая очищает содержимое таблицы, таким образом, чтобы очищалось соответствие между номерами строк таблицы и кодами отображаемых в них записей.

```

14 def clearContents(self):
15     self.__rowCode.clear()
16     QtGui.QTableWidgetItem.clearContents(self)

```

Функция «getCurrentCode» возвращает код записи в зависимости от выделенной строки таблицы.

```
18 | def getCurrentCode(self):self.__rowCode.getCode(self.currentRow())
```

В модуле «bookstable.py» определим класс «booksTable», унаследовав его от класса «dbTableWidget».

```
1 from PyQt5.QtWidgets import QTableWidgetItem
2 from dbtablewidget import dbTableWidget
3
4 class booksTable(dbTableWidget):
5     def __init__(self,library,parent=None):
6         dbTableWidget.__init__(self,library=library,header=[u'название',u'обложка',u'авторы',u'издательство',u'год',u'страницы'],parent=
7     def update(self):
8         self.clearContents()
9         self.setRowCount(len(self.getLibrary().getBookCodes()))
10        r=0
11        for a in self.getLibrary().getBookCodes():
12            self.setItem(r,0,QTableWidgetItem(self.getLibrary().getBookName(a)))
13            self.setItem(r,1,QTableWidgetItem(self.getLibrary().getBookImg(a)))
14            self.setItem(r,2,QTableWidgetItem(self.getLibrary().getBookAuthorsBibliostr(a)))
15            self.setItem(r,3,QTableWidgetItem(self.getLibrary().getBookPublName(a)))
16            self.setItem(r,4,QTableWidgetItem(str(self.getLibrary().getBookYear(a))))
17            self.setItem(r,5,QTableWidgetItem(str(self.getLibrary().getBookPages(a))))
18            self.appendRowCode(r,a)
19            r+=1
20        self.resizeColumnsToContents()
21        self.resizeRowsToContents()
22        self.setCurrentCell(0,0)
```

Определим функцию «update», которая заполняет таблицу данными из объекта «library». В строке 9 установим количество строк таблицы функцией «setRowCount» класса «QTableWidget». Количество необходимых строк получим вычислив длину (функцией «len») списка кодов книг, который можно подучить функцией «getBookCodes» класса «library», определенной при выполнении задания 2.

Заполнение ячеек таблицы происходит с помощью метода «setItem» класса «QTableWidget», который принимает на вход номер строки, номер столбца и ссылку на объект класса «QTableWidgetItem». В нашем случае, при создании объекта класса



«QTableWidgetItem» его конструктор принимает на вход текстовую строку, которую необходимо отобразить в ячейке таблицы. Данные необходимые для заполнения таблицы мы получаем при помощи методов «getBookName», «getBookAuthorsBibliostr», «getBookPublName», «getBookYear», «getBookPages» класса «library». После заполнения каждой новой строки устанавливается соответствие между номером строки и кодом отображаемой в ней записи с помощью функции «setRowCode» класса «dbTableWidget». Методы «resizeColumnsToContents», «resizeRowsToContents», «setCurrentCell» (в строках 20-21) класса «QTableWidgetItem» устанавливают оптимальный размер колонок, строк и текущую ячейку.

```

8 def update(self):
9     self.clearContents()
10    self.setRowCount(len(self.getLibrary().getBookCodes()))
11    r=0
12    for a in self.getLibrary().getBookCodes():
13        self.setItem(r,0,QTableWidgetItem(self.getLibrary().getBookName(a)))
14        self.setItem(r,1,QTableWidgetItem(self.getLibrary().getBookImg(a)))
15        self.setItem(r,2,QTableWidgetItem(self.getLibrary().getBookAuthorsBibliostr(a)))
16        self.setItem(r,3,QTableWidgetItem(self.getLibrary().getBookPublName(a)))
17        self.setItem(r,4,QTableWidgetItem(str(self.getLibrary().getBookYear(a))))
18        self.setItem(r,5,QTableWidgetItem(str(self.getLibrary().getBookPages(a))))
19        self.appendRowCode(r,a)
20        r+=1
21    self.resizeColumnsToContents()
22    self.resizeRowsToContents()
23    self.setCurrentCell(0,0)

```

Чтобы увидеть внешний вид описанной таблицы, напишем тестовую программу в модуле «test.py», которую будем в дальнейшем использовать (с небольшими изменениями) для тестирования остальных виджетов.

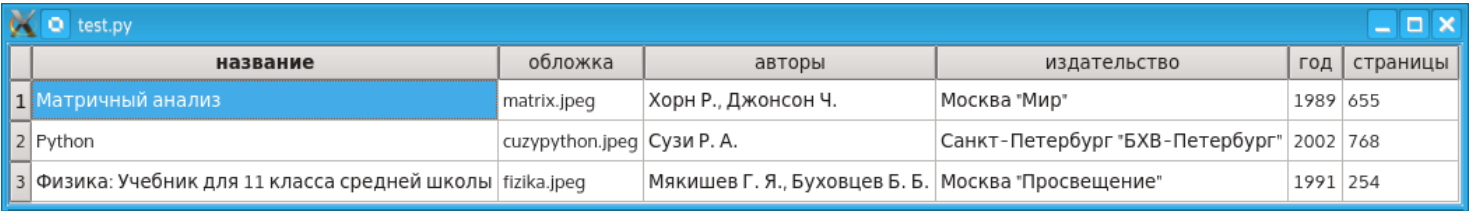
```

1 from PyQt5.QtWidgets import QApplication
2 import sys
3 sys.path.insert(0, "../library")
4 from library import library
5 from dataxml import dataxml
6 from bookstable import booksTable
7
8 app = QApplication(sys.argv)

```

```
9
10 lib=library()
11 data=dataxml()
12 data.read('old.xml',lib)
13 tw1=authorsTable(lib)
14 tw1.update()
15 tw1.show()
16
17 sys.exit(app.exec_())
18
```

Результат работы программы «test.py» приведен на рисунке (6).



	название	обложка	авторы	издательство	год	страницы
1	Матричный анализ	matrix.jpeg	Хорн Р., Джонсон Ч.	Москва "Мир"	1989	655
2	Python	cuzypython.jpeg	Сузи Р. А.	Санкт-Петербург "БХВ-Петербург"	2002	768
3	Физика: Учебник для 11 класса средней школы	fizika.jpeg	Мякишев Г. Я., Буховцев Б. Б.	Москва "Просвещение"	1991	254

Рис. 6: Таблица книг

Аналогичным образом описываем классы для просмотра списка авторов и издательств. Опишем класс «publsTable» в модуле «publstable.py».

```
1 from PyQt5.QtWidgets import QTableWidgetItem
2 from dbtablewidget import dbTableWidget
3
4 class publsTable(dbTableWidget):
5     def __init__(self,library,parent=None):
6         dbTableWidget.__init__(self,library=library,header=[u'название',u'сокр. название'],parent=parent)
7     def update(self):
8         self.clearContents()
9         self.setRowCount(len(self.getLibrary().getPublCodes()))
10        r=0
```

```

11     for a in self.getLibrary().getPublCodes():
12         self.setItem(r,0,QTableWidgetItem(self.getLibrary().getPublName(a)))
13         self.setItem(r,1,QTableWidgetItem(self.getLibrary().getPublShortname(a)))
14         self.appendRowCode(r,a)
15         r+=1
16     self.resizeColumnsToContents()
17     self.resizeRowsToContents()
18     self.setCurrentCell(0,0)

```

Опишем класс «authorsTable» в модуле «authorstable.py».

```

1  from PyQt5.QtWidgets import QTableWidgetItem
2  from dbtablewidget import dbTableWidget
3
4  class authorsTable(dbTableWidget):
5      def __init__(self,library,parent=None):
6          dbTableWidget.__init__(self,library=library,header=[u'фамилия',u'имя',u'отчество',u'сокр. имя',u'сокр. отчество'],parent=parent)
7      def update(self):
8          self.clearContents()
9          self.setRowCount(len(self.getLibrary().getAuthorCodes()))
10         r=0
11         for a in self.getLibrary().getAuthorCodes():
12             self.setItem(r,0,QTableWidgetItem(self.getLibrary().getAuthorSurname(a)))
13             self.setItem(r,1,QTableWidgetItem(self.getLibrary().getAuthorName(a)))
14             self.setItem(r,2,QTableWidgetItem(self.getLibrary().getAuthorSecname(a)))
15             self.setItem(r,3,QTableWidgetItem(self.getLibrary().getAuthorShortname(a)))
16             self.setItem(r,4,QTableWidgetItem(self.getLibrary().getAuthorShortsecname(a)))
17             self.appendRowCode(r,a)
18             r+=1
19         self.resizeColumnsToContents()
20         self.resizeRowsToContents()
21         self.setCurrentCell(0,0)

```

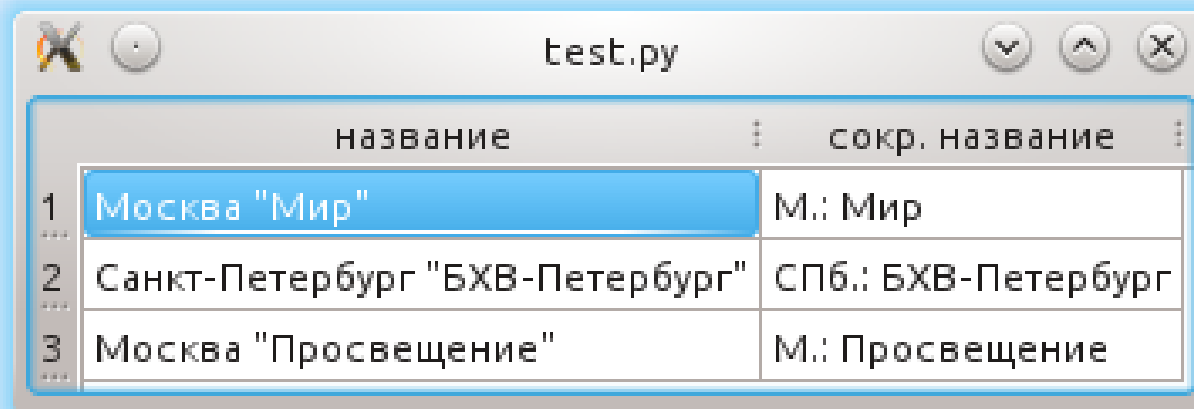
Для тестирования этих классов в модуле «test.py» в строке

```

7  from bookstable import booksTable as testwidget

```

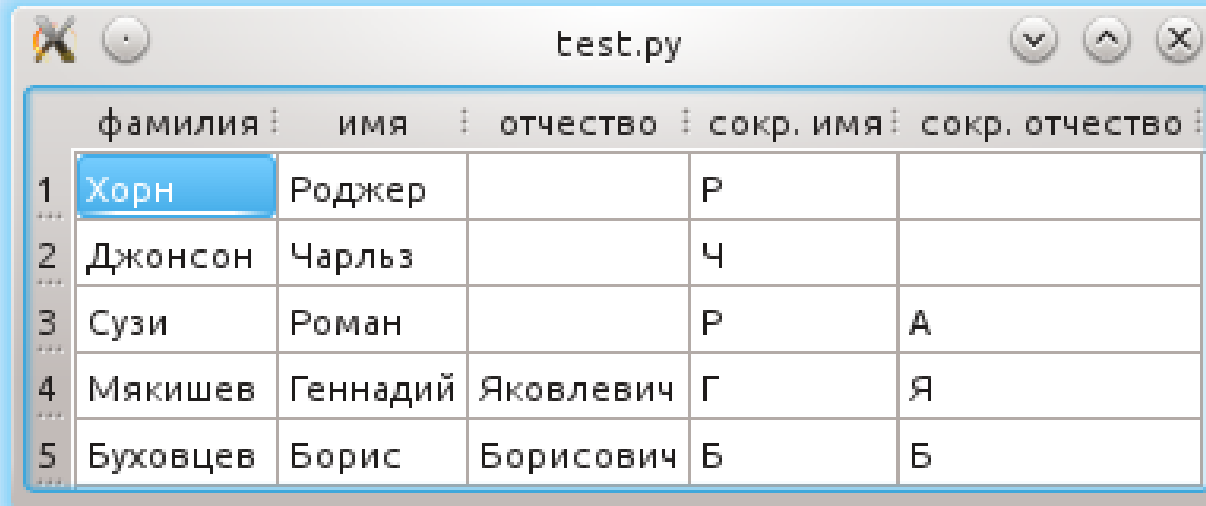
вместо названия модуля «bookstable» и класса «booksTable» вставим названия тестируемых классов и их модулей. Результаты работы программы «test.py» в этом случае приведены на рисунках (7) и (8).



The screenshot shows a window titled "test.py" with a table containing three rows of publisher information. The first row is highlighted in blue. The table has two columns: "название" (name) and "сокр. название" (abbreviated name). The rows are numbered 1, 2, and 3 on the left side of the table.

	название	сокр. название
1	Москва "Мир"	М.: Мир
2	Санкт-Петербург "БХВ-Петербург"	СПб.: БХВ-Петербург
3	Москва "Просвещение"	М.: Просвещение

Рис. 7: Таблица издательств



	фамилия	имя	отчество	сокр. имя	сокр. отчество
1	Хорн	Роджер		Р	
2	Джонсон	Чарльз		Ч	
3	Сузи	Роман		Р	А
4	Мякишев	Геннадий	Яковлевич	Г	Я
5	Буховцев	Борис	Борисович	Б	Б

Рис. 8: Таблица авторов

Создадим классы, реализующие формы для редактирования записей. Начнем с простых форм для редактирования издательств и авторов, содержащих только строки ввода «QLineEdit». Опишем класс, который будет общим предком форм для редактирования «editForm» в модуле «editform.py»

```

1 from PyQt5.QtWidgets import QWidget, QGridLayout, QVBoxLayout, QHBoxLayout, QPushButton, QLabel
2 from libwidget import libWidget
3
4 class editForm(QWidget, libWidget):
5     def __init__(self, tablewidget=None, parent=None, library=None):
6         QWidget.__init__(self, parent=parent)
7         libWidget.__init__(self, library)
8         self.__tablewidget=tablewidget

```

```

9     self.__mainvbox=QVBoxLayout()
10    self.__mainvbox.addWidget(self.__tablewidget)
11    self.__grid=QGridLayout()
12    self.__vbox=QVBoxLayout()
13    self.__hbox=QHBoxLayout()
14    self.__vbox.addLayout(self.__grid)
15    self.__vbox.addStretch(1)
16    self.__hbox.addLayout(self.__vbox)
17    self.__buttonsVBox=QVBoxLayout()
18    self.__hbox.addLayout(self.__buttonsVBox)
19    self.__newButton=QPushButton(u"Добавить")
20    self.__editButton=QPushButton(u"Изменить")
21    self.__delButton=QPushButton(u"Удалить")
22    self.__buttonsVBox.addWidget(self.__newButton)
23    self.__buttonsVBox.addWidget(self.__editButton)
24    self.__buttonsVBox.addWidget(self.__delButton)
25    self.__buttonsVBox.addStretch(1)
26    self.__mainvbox.addLayout(self.__hbox)
27    self.setLayout(self.__mainvbox)
28    self.__newButton.clicked.connect(self.newClick)
29    self.__editButton.clicked.connect(self.editClick)
30    self.__delButton.clicked.connect(self.delClick)
31    self.__tablewidget.currentCellChanged.connect(self.tableClick)
32    def getGreed(self):return self.__grid
33    def addLabel(self,text,x,y):self.__grid.addWidget(QLabel(text),x,y)
34    def addNewWidget(self,widget,x,y):self.__grid.addWidget(widget,x,y)
35    def addLeftLayout(self,layout):self.__hbox.insertLayout(0,layout)
36    def setCurrentCode(self):
37        self.__currentCode=self.__tablewidget.getCurrentCode()
38        self.update()
39    def getCurrentCode(self):return self.__currentCode
40    def decode(self,qstring):return str(qstring.toUtf8()).decode('utf-8')
41    def newClick(self):pass
42    def editClick(self):pass
43    def delClick(self):pass
44    def update(self):pass

```

```

45 def tableClick(self):self.setCurrentCode()
46 def tableUpdate(self):self.__tablewidget.update()

```

Опишем класс, реализующий форму для редактирования издательств «publEditForm» в модуле «publeditform.py», унаследовав его от класса «editForm».

```

1 from PyQt5.QtWidgets import QLineEdit
2 from editform import editForm
3 from publstable import publsTable
4
5 class publEditForm(editForm):
6     def __init__(self,parent=None,library=None):
7         editForm.__init__(self,tablewidget=publsTable(library=library),parent=parent,library=library)
8
9         self.__nameEdit=QLineEdit()
10        self.__shortnameEdit=QLineEdit()
11
12        self.addLabel(u'название',0,0)
13        self.addNewWidget(self.__nameEdit,0,1)
14        self.addLabel(u'сокр. название',1,0)
15        self.addNewWidget(self.__shortnameEdit,1,1)
16
17        self.setCurrentCode()
18
19    def update(self):
20        if self.getCurrentCode() in self.getLibrary().getPublCodes():
21            self.__nameEdit.setText(self.getLibrary().getPublName(self.getCurrentCode()))
22            self.__shortnameEdit.setText(self.getLibrary().getPublShortname(self.getCurrentCode()))
23
24    def editClick(self):
25        self.getLibrary().setPublName(self.getCurrentCode(),self.__nameEdit.text())
26        self.getLibrary().setPublShortname(self.getCurrentCode(),self.__shortnameEdit.text())
27        self.tableUpdate()
28
29    def newClick(self):
30        code=self.getLibrary().getPublNewCode()

```

```

31 |     self.getLibrary().newPubl(code)
32 |     self.getLibrary().setPublName(code,self.__nameEdit.text())
33 |     self.getLibrary().setPublShortname(code,self.__shortnameEdit.text())
34 |     self.tableUpdate()
35 |
36 | def delClick(self):
37 |     self.getLibrary().removePubl(self.getCurrentCode())
38 |     self.tableUpdate()

```

Для тестирования этого класса изменим модуль «test.py» следующим образом. В строке

```

7 | from bookstable import booksTable as testwidget

```

изменим название тестируемого класса и соответствующего модуля. В строке 16

```

15 | tw=testwidget(lib)
16 | tw.update()
17 | tw.show()

```

заменим метод «update()» на «setCurrentCode(1)»:

```

15 | tw=testwidget(lib)
16 | tw.setCurrentCode(1)
17 | tw.show()

```

Результат работы программы «test.py» в этом случае приведен на рисунке (9).



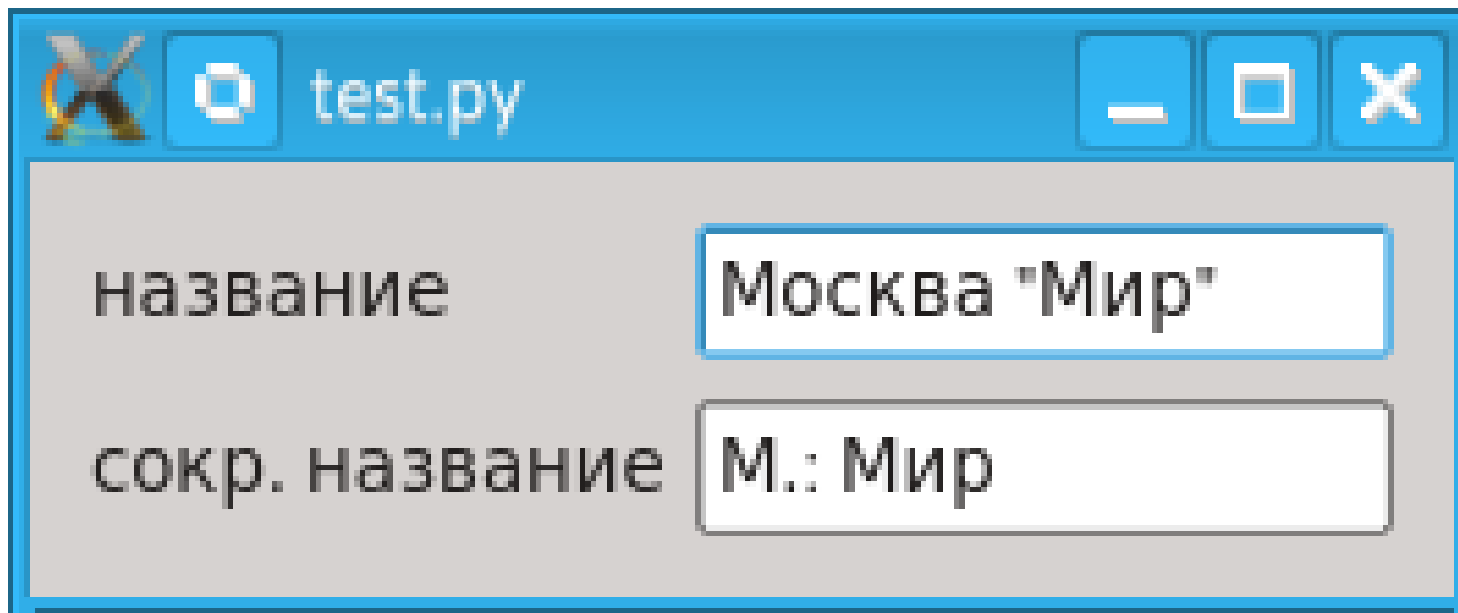


Рис. 9: Форма для редактирования издательств

Опишем класс, реализующий форму для редактирования авторов «authorEditForm» в модуле «authoreditform.py», унаследовав его от класса «editForm».

```

1 from PyQt5.QtWidgets import QLineEdit
2 from editform import editForm
3 from authorstable import authorsTable
4
5 class authorEditForm(editForm):
6     def __init__(self, parent=None, library=None):
7         editForm.__init__(self, tablewidget=authorsTable(library=library), parent=parent, library=library)
8
9         self.__surnameEdit=QLineEdit()
10        self.__nameEdit=QLineEdit()
11        self.__secnameEdit=QLineEdit()
12        self.__shortnameEdit=QLineEdit()
13        self.__shortsecnameEdit=QLineEdit()

```

```

14 self.addLabel('фамилия',0,0)
15 self.addNewWidget(self.__surnameEdit,0,1)
16 self.addLabel('имя',1,0)
17 self.addNewWidget(self.__nameEdit,1,1)
18 self.addLabel('отчество',2,0)
19 self.addNewWidget(self.__secnameEdit,2,1)
20 self.addLabel('сокр. имя',3,0)
21 self.addNewWidget(self.__shortnameEdit,3,1)
22 self.addLabel('сокр. отчество',4,0)
23 self.addNewWidget(self.__shortsecnameEdit,4,1)
24
25 self.setCurrentCode()
26
27
28 def update(self):
29     if self.getCurrentCode() in self.getLibrary().getAuthorCodes():
30         self.__surnameEdit.setText(self.getLibrary().getAuthorSurname(self.getCurrentCode()))
31         self.__nameEdit.setText(self.getLibrary().getAuthorName(self.getCurrentCode()))
32         self.__secnameEdit.setText(self.getLibrary().getAuthorSecname(self.getCurrentCode()))
33         self.__shortnameEdit.setText(self.getLibrary().getAuthorShortname(self.getCurrentCode()))
34         self.__shortsecnameEdit.setText(self.getLibrary().getAuthorShortsecname(self.getCurrentCode()))
35
36 def editClick(self):
37     self.getLibrary().setAuthorSurname(self.getCurrentCode(),self.__surnameEdit.text())
38     self.getLibrary().setAuthorName(self.getCurrentCode(),self.__nameEdit.text())
39     self.getLibrary().setAuthorSecname(self.getCurrentCode(),self.__secnameEdit.text())
40     self.getLibrary().setAuthorShortname(self.getCurrentCode(),self.__shortnameEdit.text())
41     self.getLibrary().setAuthorShortsecname(self.getCurrentCode(),self.__shortsecnameEdit.text())
42     self.tableUpdate()
43
44 def newClick(self):
45     code=self.getLibrary().getAuthorNewCode()
46     self.getLibrary().newAuthor(code)
47     self.getLibrary().setAuthorSurname(code,self.__surnameEdit.text())
48     self.getLibrary().setAuthorName(code,self.__nameEdit.text())
49     self.getLibrary().setAuthorSecname(code,self.__secnameEdit.text())

```

```
50     self.getLibrary().setAuthorShortname(code,self.__shortnameEdit.text())
51     self.getLibrary().setAuthorShortsecname(code,self.__shortsecnameEdit.text())
52     self.tableUpdate()
53
54 def delClick(self):
55     self.getLibrary().removeAuthor(self.getCurrentCode())
56     self.tableUpdate()
```

Протестируем аналогично форме для редактирования издательств, заменив название тестируемого класса и соответствующего модуля. Результат работы программы «test.py» в этом случае приведен на рисунке (10).

фамилия Хорн|

имя Роджер

отчество

сокр. имя Р

сокр. отчество

Рис. 10: Форма для редактирования авторов

Прежде чем создать форму для редактирования книг нам потребуется описать несколько новых виджетов – список авторов книги (`QListWidget`), с помощью которого можно удалять авторов из книги, поле со списком (`QComboBox`) авторов,

чтобы добавлять авторов в книгу, поле со списком издательств, чтобы менять издательство книги. Для полей со списком стоит описать класс-предок, так как у них много общего. Аналогично поступим со списком авторов (на всякий случай). От стандартных классов QT QListWidget и QComboBox наши классы отличаются наличием в своей структуре объекта класса «rowCode» для установления связи между строками списков и кодами соответствующих записей.

В модуле «dblistwidget.py» опишем класс «dbListWidget», являющийся предком класса «authorListWidget».

```
1 from PyQt5.QtWidgets import QListWidget
2 from rowcode import rowCode
3 from libwidget import libWidget
4
5 class dbListWidget(QListWidget,libWidget):
6     def __init__(self,parent=None,library=None):
7         QListWidget.__init__(self,parent)
8         libWidget.__init__(self,library)
9         self.__rowCode=rowCode()
10    def clear(self):
11        self.__rowCode.clear()
12        QListWidget.clear(self)
13    def addItem(self,code,text):
14        self.__rowCode.appendRowCode(self.count(),code)
15        QListWidget.addItem(self,text)
16    def removeSelected(self):
17        self.__rowCode.removeRow(self.currentRow())
18        for item in self.selectedItems():
19            self.takeItem(self.row(item))
20    def getCurrentCode(self):return self.__rowCode.getCode(self.currentRow())
21    def setCurrentRec(self,value):
22        self.__currentRec=value
23        self.update()
24    def getCurrentRec(self):return self.__currentRec
25    def getCodes(self):return self.__rowCode.getCodes()
26    def update(self):pass
```

В модуле «authorlistwidget.py» опишем класс «authorListWidget» для отображения списка авторов книги, наследуемый от класса «dbListWidget».

```

1 from dblistwidget import dbListWidget
2
3 class authorListWidget(dbListWidget):
4     def update(self):
5         self.clear()
6         for a in self.getLibrary().getBookAuthorCodes(self.getCurrentRec()):
7             self.addItem(a,self.getLibrary().getAuthorBibliostr(a))
8         if self.getLibrary().getBookAuthorCodes(self.getCurrentRec()):self.setCurrentRow(0)

```

В модуле «dbcombobox.py» опишем класс «dbComboBox», являющийся предком классов «authorCombo» и «publCombo».

```

1 from PyQt5.QtWidgets import QComboBox
2 from rowcode import rowCode
3 from libwidget import libWidget
4
5 class dbComboBox(QComboBox,libWidget):
6     def __init__(self,parent=None,library=None):
7         QComboBox.__init__(self,parent)
8         libWidget.__init__(self,library)
9         self.__rowCode=rowCode()
10        self.setSizeAdjustPolicy(self.AdjustToContents)
11    def clear(self):
12        self.__rowCode.clear()
13        QComboBox.clear(self)
14    def addItem(self,code,text):
15        self.__rowCode.appendRowCode(self.count(),code)
16        QComboBox.addItem(self,text)
17    def removeItem(self,index):
18        self.__rowCode.removeRow(index)
19        QComboBox.removeItem(self,index)
20    def getCurrentCode(self):return self.__rowCode.getCode(self.currentIndex())
21    def setCurrentCode(self,code):
22        if self.__rowCode.getRow(code):self.setCurrentIndex(self.__rowCode.getRow(code))
23    def setCurrentRec(self,value):
24        self.__currentRec=value
25        self.update()

```

```

26 | def getCurrentRec(self):return self.__currentRec
27 | def update(self):pass

```

В модуле «authorcombo.py» опишем класс «authorCombo» для выбора авторов, которые могут быть добавлены в книгу, наследуемый от класса «dbComboBox».

```

1 | from dbcombobox import dbComboBox
2 |
3 | class authorCombo(dbComboBox):
4 |     def update(self):
5 |         self.clear()
6 |         for a in self.getLibrary().getAuthorCodes():
7 |             if not(a in self.getLibrary().getBookAuthorCodes(self.getCurrentRec())):
8 |                 self.addItem(a,self.getLibrary().getAuthorBibliostr(a))

```

В модуле «publcombo.py» опишем класс «publCombo» для отображения и изменения издательства книги, наследуемый от класса «dbComboBox».

```

1 | from dbcombobox import dbComboBox
2 |
3 | class publCombo(dbComboBox):
4 |     def update(self):
5 |         self.clear()
6 |         for p in self.getLibrary().getPublCodes():
7 |             self.addItem(p,self.getLibrary().getPublName(p))
8 |         self.setCurrentCode(self.getLibrary().getBookPublCode(self.getCurrentRec()))

```

Опишем класс, реализующий форму для редактирования книг «bookEditForm» в модуле «bookeditform.py», унаследовав его от класса «editForm».

```

1 | from PyQt5.QtWidgets import QVBoxLayout,QLineEdit,QPushButton,QLabel,QSpinBox,QFileDialog
2 | from PyQt5 import QtCore
3 | from PyQt5.QtGui import QPixmap
4 | from editform import editForm

```

```

5 from dblistwidget import dbListWidget
6 from dbcombobox import dbComboBox
7 from publcombo import publCombo
8 from authorcombo import authorCombo
9 from authorlistwidget import authorListWidget
10 from bookstable import booksTable
11
12 class bookEditForm(editForm):
13     def __init__(self, parent=None, library=None):
14         editForm.__init__(self, tablewidget=booksTable(library=library), parent=parent, library=library)
15
16         self.__pixlabel=QLabel()
17         self.__nameEdit=QLineEdit()
18         self.__imgEdit=QLineEdit()
19         self.__imgButton=QPushButton(u'найти')
20         self.__authorListWidget=authorListWidget(library=library)
21         self.__removeButton=QPushButton(u'удалить')
22         self.__authorCombo=authorCombo(library=library)
23         self.__appendButton=QPushButton(u'добавить')
24         self.__publCombo=publCombo(library=library)
25         self.__yearSpin=QSpinBox()
26         self.__yearSpin.setRange(-3000,QtCore.QDate().currentDate().year())
27         self.__pagesSpin=QSpinBox()
28         self.__pagesSpin.setRange(0,10000)
29
30         self.addLabel('название',0,0)
31         self.addNewWidget(self.__nameEdit,0,1)
32         self.addLabel(u'обложка',1,0)
33         self.addNewWidget(self.__imgEdit,1,1)
34         self.addNewWidget(self.__imgButton,1,2)
35         self.addLabel(u'авторы',2,0)
36         self.addNewWidget(self.__authorListWidget,2,1)
37         self.addNewWidget(self.__removeButton,2,2)
38         self.addNewWidget(self.__authorCombo,3,1)
39         self.addNewWidget(self.__appendButton,3,2)
40         self.addLabel(u'издательство',4,0)

```



```

41 self.addNewWidget(self.__publCombo,4,1)
42 self.addLabel(u'год',5,0)
43 self.addNewWidget(self.__yearSpin,5,1)
44 self.addLabel(u'страницы',6,0)
45 self.addNewWidget(self.__pagesSpin,6,1)
46
47 self.__pixVBox=QVBoxLayout()
48 self.__pixVBox.addWidget(self.__pixlabel)
49 self.__pixVBox.addStretch(1)
50 self.addLeftLayout(self.__pixVBox)
51
52 self.__removeButton.clicked.connect(self.removeAuthor)
53 self.__appendButton.clicked.connect(self.appendAuthor)
54 self.__imgButton.clicked.connect(self.openImg)
55
56 self.setCurrentCode()
57
58 def openImg(self):
59     filename=QFileDialog.getOpenFileName(self, 'Open file', './')[0]
60     if filename:
61         self.__imgEdit.setText(filename)
62         self.__pixlabel.setPixmap(QPixmap(filename))
63
64 def update(self):
65     if self.getCurrentCode() in self.getLibrary().getBookCodes():
66         self.__nameEdit.setText(self.getLibrary().getBookName(self.getCurrentCode()))
67         self.__imgEdit.setText(self.getLibrary().getBookImg(self.getCurrentCode()))
68         self.__pixlabel.setPixmap(QPixmap(self.getLibrary().getBookImg(self.getCurrentCode())))
69         self.__publCombo.setCurrentRec(self.getCurrentCode())
70         self.__authorListWidget.setCurrentRec(self.getCurrentCode())
71         self.__authorCombo.setCurrentRec(self.getCurrentCode())
72         self.__yearSpin.setValue(self.getLibrary().getBookYear(self.getCurrentCode()))
73         self.__pagesSpin.setValue(self.getLibrary().getBookPages(self.getCurrentCode()))
74
75 def removeAuthor(self):
76     code=self.authorListWidget.getCurrentCode()

```

```

77     if code:
78         self.__authorListWidget.removeSelected()
79         self.__authorCombo.addItem(code,self.getLibrary().getAuthorBibliostr(code))
80
81 def appendAuthor(self):
82     code=self.__authorCombo.getCurrentCode()
83     if code:
84         self.__authorCombo.removeItem(self.__authorCombo.currentIndex())
85         self.__authorListWidget.addItem(code,self.getLibrary().getAuthorBibliostr(code))
86
87 def editClick(self):
88     self.getLibrary().setBookName(self.getCurrentCode(),self.__nameEdit.text())
89     self.getLibrary().setBookImg(self.getCurrentCode(),self.__imgEdit.text())
90     self.getLibrary().clearBookAuthors(self.getCurrentCode())
91     for c in self.__authorListWidget.getCodes():
92         self.getLibrary().appendBookAuthor(self.getCurrentCode(),self.getLibrary().findAuthorByCode(c))
93     self.getLibrary().setBookPubl(self.getCurrentCode(),self.__publCombo.getCurrentCode())
94     self.getLibrary().setBookYear(self.getCurrentCode(),self.__yearSpin.value())
95     self.getLibrary().setBookPages(self.getCurrentCode(),self.__pagesSpin.value())
96     self.tableUpdate()
97
98 def newClick(self):
99     code=self.getLibrary().getBookNewCode()
100    self.getLibrary().newBook(code)
101    self.getLibrary().setBookName(code,self.__nameEdit.text())
102    self.getLibrary().setBookImg(code,self.__imgEdit.text())
103    for c in self.__authorListWidget.getCodes():
104        self.getLibrary().appendBookAuthor(code,self.getLibrary().findAuthorByCode(c))
105    self.getLibrary().setBookPubl(code,self.__publCombo.getCurrentCode())
106    self.getLibrary().setBookYear(code,self.__yearSpin.value())
107    self.getLibrary().setBookPages(code,self.__pagesSpin.value())
108    self.tableUpdate()
109
110 def delClick(self):
111     self.getLibrary().removeBook(self.getCurrentCode())
112     self.tableUpdate()

```

Протестируем полученный класс с помощью программы «test.py», заменив название тестируемого класса и соответствующего модуля. Результат работы программы «test.py» в этом случае приведен на рисунке (11).

The screenshot shows a graphical user interface for editing book information. On the left is a book cover for 'МАТРИЧНЫЙ АНАЛИЗ' by Р. Хорн, Ч. Джонсон, published by Издательство «Мир». The right side contains a form with the following fields and controls:

- название**: Text input field containing 'Матричный анализ'.
- обложка**: Text input field containing 'matrix.jpeg', with a 'найти' (find) button to its right.
- авторы**: A list box containing 'Хорн Р.' and 'Джонсон Ч.', with a 'удалить' (delete) button to its right.
- издательство**: A dropdown menu currently showing 'Сузи Р. А.', with a 'добавить' (add) button to its right.
- год**: A text input field containing '1989'.
- страницы**: A text input field containing '655'.

Рис. 11: Форма для редактирования книг

В модуле «tab.py» опишем класс «tabWidget» располагающий страницы для редактирования сущностей на форме с вкладками.

```

1 from PyQt5.QtWidgets import QTabWidget
2 import sys,os
3 from bookeditform import bookEditForm
4 from authoreditform import authorEditForm
5 from publeditform import publEditForm

```

```

6
7 class tabWidget(QTabWidget):
8     def __init__(self, library, parent=None):
9         QTabWidget.__init__(self, parent)
10        self.__bookEditForm=bookEditForm(library=library)
11        self.addTab(self.__bookEditForm, u"книги")
12        self.__authorEditForm=authorEditForm(library=library)
13        self.addTab(self.__authorEditForm, u"авторы")
14        self.__publEditForm=publEditForm(library=library)
15        self.addTab(self.__publEditForm, u"издательства")
16    def update(self):
17        self.__publEditForm.tableUpdate()
18        self.__authorEditForm.tableUpdate()
19        self.__bookEditForm.tableUpdate()

```

Наконец реализуем главное окно программы с возможностью создания новой базы данных, загрузки, сохранения в форматах XML и sqlite3, в модкле «main4.py»

```

1 from PyQt5.QtWidgets import QApplication, QMainWindow, QAction, QFileDialog
2 from PyQt5.QtGui import QIcon
3 import sys, os
4 sys.path.insert(0, "./library")
5 from datasql import datasql
6 from dataxml import dataxml
7 from library import library
8 from tab import tabWidget
9
10 app = QApplication(sys.argv)
11
12 class mainWindow(QMainWindow):
13     def __init__(self):
14         QMainWindow.__init__(self)
15         self.setWindowTitle(u'Библиотека')
16         self.library=library()
17         self.dataxml=dataxml()
18         self.datasql=datasql()

```

```

19 self.tabWidget=tabWidget(self.library,self)
20 self.setCentralWidget(self.tabWidget)
21 self.tabWidget.update()
22
23 self.new=QAction(QIcon(),'New',self)
24 self.new.setStatusTip('New database')
25 self.new.triggered.connect(self.newAction)
26
27 self.openxml=QAction(QIcon(),'Open XML',self)
28 self.openxml.setStatusTip('Open data from XML')
29 self.openxml.triggered.connect(self.openXMLAction)
30
31 self.opensql=QAction(QIcon(),'Open SQL',self)
32 self.opensql.setStatusTip('Open data from SQL')
33 self.opensql.triggered.connect(self.openSQLAction)
34
35 self.savexml=QAction(QIcon(),'Save XML',self)
36 self.savexml.setStatusTip('Save data to XML')
37 self.savexml.triggered.connect(self.saveXMLAction)
38
39 self.savesql=QAction(QIcon(),'Save SQL',self)
40 self.savesql.setStatusTip('Save data to SQL')
41 self.savesql.triggered.connect(self.saveSQLAction)
42
43 self.menubar=self.menuBar()
44 self.menufile=self.menubar.addMenu('&File')
45 self.menufile.addAction(self.new)
46 self.menufile.addSeparator()
47 self.menufile.addAction(self.openxml)
48 self.menufile.addAction(self.opensql)
49 self.menufile.addSeparator()
50 self.menufile.addAction(self.savexml)
51 self.menufile.addAction(self.savesql)
52
53 self.statusBar()
54

```

```

55 def newAction(self):
56     self.library.clear()
57     self.tabWidget.update()
58
59 def openXMLAction(self):
60     filename=QFileDialog.getOpenFileName(self,u'Открыть XML',os.getcwd(),u"*.xml")[0]
61     if filename:
62         self.library.clear()
63         self.dataxml.read(filename,self.library)
64         self.tabWidget.update()
65
66 def openSQLAction(self):
67     filename=QFileDialog.getOpenFileName(self,u'Открыть SQL',os.getcwd(),u"*.sqlite")[0]
68     if filename:
69         self.library.clear()
70         self.datasql.read(filename,self.library)
71         self.tabWidget.update()
72
73 def saveXMLAction(self):
74     filename=QFileDialog.getSaveFileName(self,u'Сохранить XML',os.getcwd(),u"*.xml")[0]
75     if filename:self.dataxml.write(filename,self.library)
76
77 def saveSQLAction(self):
78     filename=QFileDialog.getSaveFileName(self,u'Сохранить SQL',os.getcwd(),u"*.sqlite")[0]
79     if filename:self.datasql.write(filename,self.library)
80
81 mw=mainWindow()
82 mw.show()
83 sys.exit(app.exec_())

```

В результате выполнения программы из модуля «main4.py» получим окно вид которого приведен на рисунке (5).

## 6.1 Виджеты для работы с датами

В случае, когда в базе данных необходимо хранить сведения о датах можно использовать виджеты «QCalendarWidget» и «QDateEdit». Ниже приведен пример использования этих виджетов и конвертации даты в текст и обратно.

```
1 import sys
2 from PyQt5 import QtCore
3 from PyQt5.QtWidgets import QApplication, QWidget, QCalendarWidget, QVBoxLayout, QLineEdit, QDateEdit, QPushButton, QHBoxLayout
4
5 class Calendar(QWidget):
6     def __init__(self, parent=None):
7         QWidget.__init__(self, parent)
8         self.vbox = QVBoxLayout()
9         self.cal = QCalendarWidget(self)
10        self.cal.setGridVisible(True)
11        self.edit=QLineEdit(self)
12        self.datedit=QDateEdit(self)
13        self.datedit.setDisplayFormat('yyyy-MM-dd')
14        self.datedit.setCalendarWidget(QCalendarWidget())
15        self.datedit.setCalendarPopup(True)
16        self.but=QPushButton(self)
17        self.vbox.addWidget(self.cal)
18        self.vbox.addWidget(self.datedit)
19        self.hbox = QHBoxLayout()
20        self.vbox.addLayout(self.hbox)
21        self.hbox.addWidget(self.edit)
22        self.hbox.addWidget(self.but)
23        self.setLayout(self.vbox)
24
25        self.cal.selectionChanged.connect(self.settextlinedate)
26        self.but.clicked.connect(self.setcalendardate)
27
28    def settextlinedate(self):
29        self.edit.setText(self.cal.selectedDate().toString('yyyy-MM-dd'))
30        self.datedit.setDate(self.cal.selectedDate())
31    def setcalendardate(self):
```

```

32     self.cal.setSelectedDate(QtCore.QDate.fromString(self.edit.text(), 'yyyy-MM-dd'))
33     self.cal.setSelectedDate(self.datedit.date())
34
35 app = QApplication(sys.argv)
36 form = Calendar()
37 form.show()
38 app.exec_()

```

В результате работы программы на экране появится окно, приведенное на рисунке (12).



Рис. 12: Пример использования виджетов для работы с датой

В данном приложении в случае выбора даты в календаре (QCalendarWidget) устанавливается соответствующее значение даты в строке ввода даты (QDateEdit) и в строке ввода текста (QLineEdit). Строка ввода даты отличается возможностью



листать выделенный элемент даты (в данном случае год-месяц-число) с помощью кнопок вверх/вниз, как в QSpinBox. При нажатии на кнопку внизу формы, дата заданная в виде текста в строке ввода текста преобразуется в дату и устанавливается в календаре.

## 6.2 Использование MVC

Пример списочной модели

```
1 from PyQt4 import QtGui,QtCore
2 import sys
3
4 app=QtGui.QApplication(sys.argv)
5
6 data=QtCore.QStringList(["one","two","three","four","five"])
7
8 listWidget=QtGui.QListWidget()
9 listWidget.show()
10 listWidget.addItem(data)
11
12 count=listWidget.count()
13 for i in range(count):
14     item=listWidget.item(i)
15     item.setFlags(item.flags() | QtCore.Qt.ItemIsEditable)
16
17 comboBox=QtGui.QComboBox()
18 comboBox.show()
19 comboBox.addItem(data)
20
21 sys.exit(app.exec_())
```

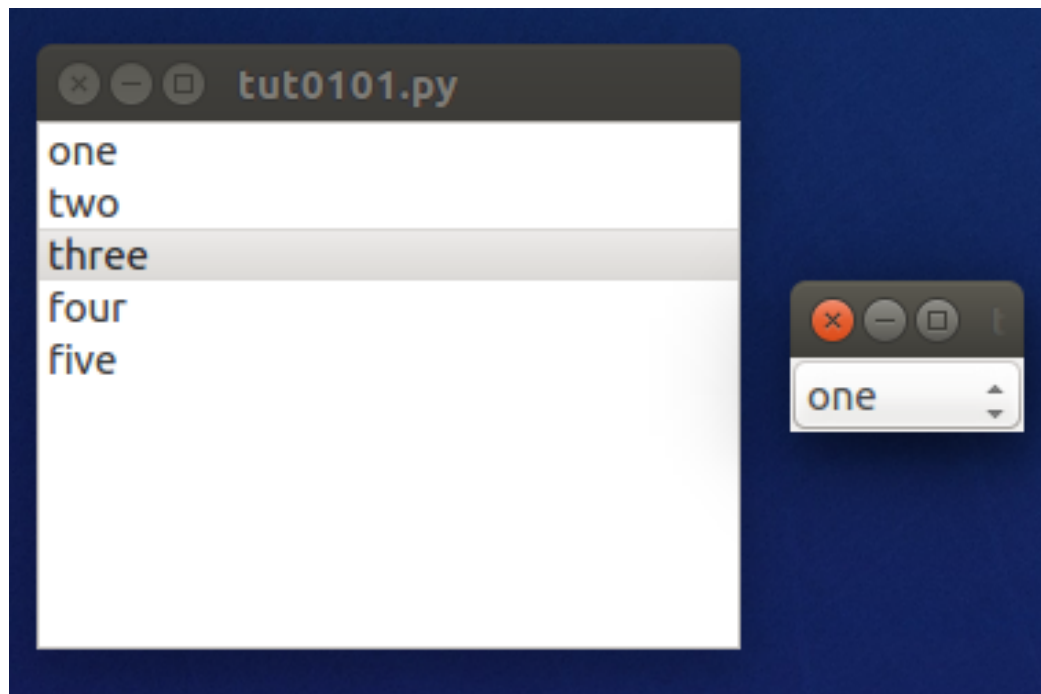


Рис. 13: QStringList

Пример нескольких представлений одной модели

```
1  -*- coding:utf-8 -*-
2  from PyQt4 import QtGui,QtCore
3  import sys
4
5  class mainWin(QtGui.QWidget):
6      def __init__(self,parent=None):
7          QtGui.QWidget.__init__(self,parent)
8
9          self.listView=QtGui.QListView()
10         self.treeView=QtGui.QTreeView()
11         self.tableView=QtGui.QTableView()
12         self.comboBox=QtGui.QComboBox()
```

```

13
14     self.grid=QtGui.QGridLayout()
15     self.setLayout(self.grid)
16     self.grid.addWidget(self.listView,0,0)
17     self.grid.addWidget(self.treeView,0,1)
18     self.grid.addWidget(self.tableView,1,0)
19     self.grid.addWidget(self.comboBox,1,1)
20
21     red=QtGui.QColor(255,0,0)
22     green=QtGui.QColor(0,255,0)
23     blue=QtGui.QColor(0,0,255)
24
25     self.model=PaletteListModel([red,green,blue])
26
27     self.listView.setModel(self.model)
28     self.treeView.setModel(self.model)
29     self.tableView.setModel(self.model)
30     self.comboBox.setModel(self.model)
31
32 class PaletteListModel(QtCore.QAbstractListModel):
33     def __init__(self,colors=[],parent=None):
34         QtCore.QAbstractListModel.__init__(self,parent)
35         self.__colors=colors
36
37     def rowCount(self,parent):
38         return len(self.__colors)
39
40 # отображение данных
41     def data(self,index,role):
42         if role==QtCore.Qt.DisplayRole:
43             return self.__colors[index.row()].name()
44         if role==QtCore.Qt.DecorationRole:
45             pixmap=QtGui.QPixmap(26,26)
46             pixmap.fill(self.__colors[index.row()])
47             icon=QtGui.QIcon(pixmap)
48             return icon

```

```

49     if role==QtCore.Qt.ToolTipRole:
50         return "Hex code: "+self.__colors[index.row()].name()
51     if role==QtCore.Qt.EditRole:
52         return self.__colors[index.row()].name()
53
54 # установка заголовков
55 def headerData(self,section,orientation,role):
56     if role==QtCore.Qt.DisplayRole:
57         if orientation==QtCore.Qt.Horizontal:return QtCore.QString("Palette")
58         else: return QtCore.QString("Color %1").arg(section)
59
60 # разрешение на редактирование данных
61 def flags(self,index):
62     return QtCore.Qt.ItemIsEditable | QtCore.Qt.ItemIsEnabled | QtCore.Qt.ItemIsSelectable
63
64 # редактирование данных
65 def setData(self,index,value,role=QtCore.Qt.EditRole):
66     if role==QtCore.Qt.EditRole:
67         row=index.row()
68         color=QtGui.QColor(value)
69         if color.isValid():
70             self.__colors[row]=color
71             self.dataChanged.emit(index,index)
72             return True
73     return False
74
75 # добавление данных
76 def insertRows(self,position,rows,parent=QtCore.QModelIndex()):
77     self.beginInsertRows(parent,position,position+rows-1)
78     for i in range(rows):
79         self.__colors.insert(position,QtGui.QColor("#000000"))
80     self.endInsertRows()
81
82 # удаление данных
83 def removeRows(self,position,rows,parent=QtCore.QModelIndex()):
84     self.beginRemoveRows(parent,position,position+rows-1)

```

```
85     for i in range(rows):
86         value=self.__colors[position]
87         self.__colors.remove(value)
88     self.endRemoveRows()
89
90 app=QtGui.QApplication(sys.argv)
91
92 mw=mainWin()
93 mw.show()
94 mw.model.insertRows(0,5)
95 mw.model.removeRows(1,4)
96
97 sys.exit(app.exec_())
```

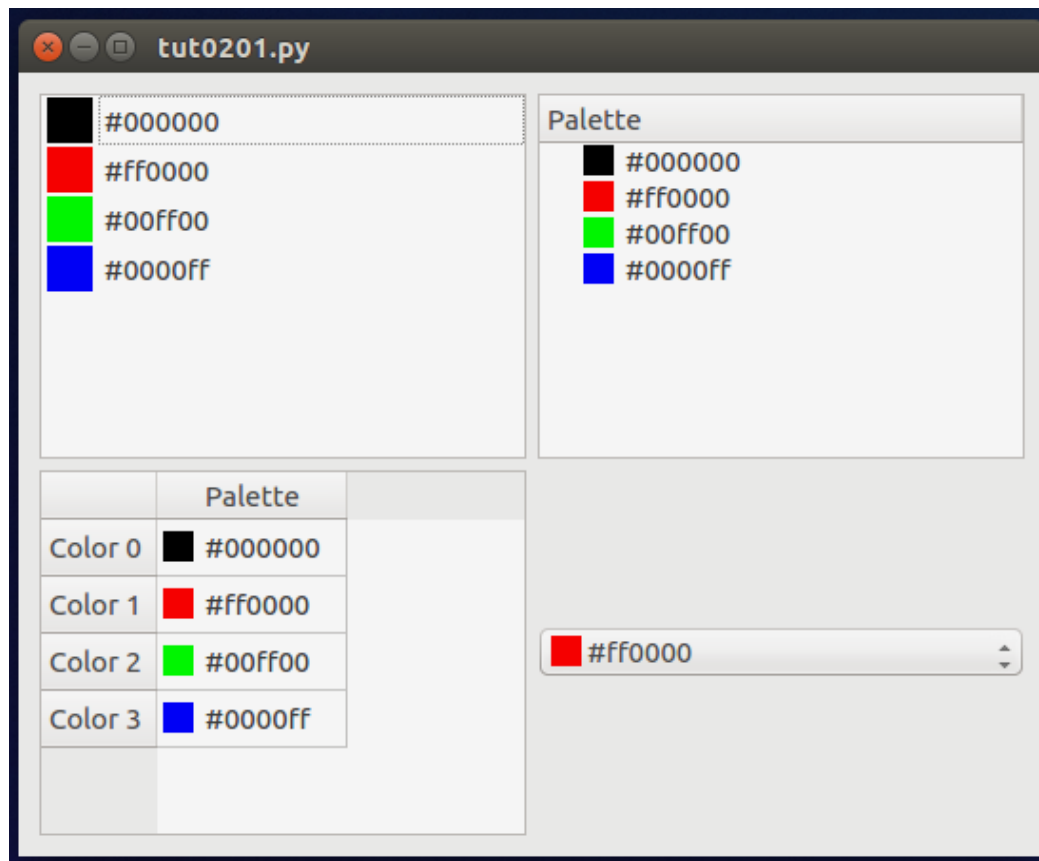


Рис. 14: PaletteListModel

## Работа с базой данных

```

1 from PyQt4 import QtGui,QtCore,QtSql
2 import sys
3
4 def decode(qstring):
5     return str(qstring.toUtf8()).decode('utf-8')
6
7 app = QtGui.QApplication(sys.argv)
8

```

```

9 dbase=QtSql.QSqlDatabase.addDatabase('SQLITE')
10 dbase.setDatabaseName(u'autoprokat.sqlite')
11 dbase.open()
12 query=QtSql.QSqlQuery(dbase)
13 query.exec_('''
14     SELECT
15         zakaz.kod,
16         client.fio,
17         car.nomer,
18         zakaz.date1,
19         zakaz.date2,
20         zakaz.vozvrat
21     FROM
22         (zakaz inner join client on zakaz.client=client.kod)
23         inner join car on zakaz.car=car.kod''')
24
25 query.first()
26 while query.isValid():
27     s= '|' +str(query.value(0).toString().toUtf8()).decode('utf-8')+'|'
28     s+= str(query.value(1).toString().toUtf8()).decode('utf-8')+'|'
29     s+= str(query.value(2).toString().toUtf8()).decode('utf-8')+'|'
30     s+= str(query.value(3).toString().toUtf8()).decode('utf-8')+'|'
31     s+= str(query.value(4).toString().toUtf8()).decode('utf-8')+'|'
32     s+= str(query.value(5).toString().toUtf8()).decode('utf-8')+'|'
33     print s
34     query.next()
35
36 mw = mainwin()
37 mw.show()
38
39 sys.exit(app.exec_())

```

1	Сергеева Наталья Александровна	ф123ке	2000-01-01	15.04.16	15.04.16
2	Мамедов Андрей Васильевич	м542ро	2012-06-21	10.06.16	10.06.16
3	Зверева Вера Ивановна	р654та	2000-01-01	30.06.16	30.06.16
4	Мамедов Андрей Васильевич	м542ро	11.06.16	15.06.16	15.07.16
5	Куликов Игорь Иванович	ы985рп	2008-01-01	10.06.16	10.06.16

Рис. 15: Работа с базой данных

## Использование делегатов

```

1 from PyQt4 import QtGui,QtCore,QtSql
2 import sys
3
4 def decode(qstring):
5     return str(qstring.toUtf8()).decode('utf-8')
6
7 class dateEditDelegate(QtGui.QItemDelegate):
8     def __init__(self, parent=None):
9         #QtGui.QItemDelegate.__init__(parent)
10        super(dateEditDelegate, self).__init__(parent)
11
12    def createEditor(self, parent, option, index):
13        editor=QtGui.QDateEdit(parent)
14        editor.setDisplayFormat('yyyy-MM-dd')
15        editor.setCalendarWidget(QtGui.QCalendarWidget())
16        editor.setCalendarPopup(True)
17        return editor
18
19    def setEditorData(self,editor,index):
20        value=index.model().data(index,QtCore.Qt.EditRole)
21        editor.setDate(QtCore.QDate.fromString(value.toString(),'yyyy-MM-dd'))
22        if index.column()==3:
23            editor.setMaximumDate(QtCore.QDate.fromString(index.model.record(index.row()).value(4),'yyyy-MM-dd'))
24        if index.column()==4:
25            editor.setMinimumDate(QtCore.QDate.fromString(model.record(index.row()).value(3),'yyyy-MM-dd'))
26        if index.column()==5:
27            editor.setMinimumDate(QtCore.QDate.fromString(model.record(index.row()).value(3),'yyyy-MM-dd'))

```



```

28
29     def setModelData(self, editor, model, index):
30         value=editor.date().toString('yyyy-MM-dd')
31         model.setData(index, value, QtCore.Qt.EditRole)
32
33 class mainwin(QtGui.QWidget):
34     def __init__(self, parent=None):
35         QtGui.QWidget.__init__(self, parent)
36         QtCore.QLocale.setDefault(QtCore.QLocale('ru_RU'))
37         self.mainlayout=QtGui.QVBoxLayout(self)
38         self.setLayout(self.mainlayout)
39
40         self.dbase=QtSql.QSqlDatabase.addDatabase('SQLITE')
41         self.dbase.setDatabaseName(u'autoprokat.sqlite')
42         self.dbase.open()
43
44         self.model=QtSql.QSqlRelationalTableModel(self, self.dbase)
45         self.model.setTable('zakaz')
46         self.model.setRelation(1, QSql.QSqlRelation("client", "kod", "fio"))
47         self.model.setRelation(2, QSql.QSqlRelation("car", "kod", "nomer"))
48         self.model.select()
49
50         self.table=QtGui.QTableView(self)
51         self.mainlayout.addWidget(self.table)
52         self.table.setModel(self.model)
53         self.table.setAlternatingRowColors(True)
54         self.table.setSortingEnabled(True)
55         self.table.setItemDelegate(QSql.QSqlRelationalDelegate(self.table))
56         self.table.setItemDelegateForColumn(3, dateEditDelegate(self.table))
57         self.table.setItemDelegateForColumn(4, dateEditDelegate(self.table))
58         self.table.setItemDelegateForColumn(5, dateEditDelegate(self.table))
59
60 app = QtGui.QApplication(sys.argv)
61
62 mw = mainwin()
63 mw.show()

```

```
64  
65 sys.exit(app.exec_())
```

## Использование QTreeWidget

```
1 from PyQt4 import QtCore, QtGui  
2 import sys  
3  
4 app = QtGui.QApplication(sys.argv)  
5 QtGui.qApp = app  
6  
7 folderTree = QtGui.QTreeWidget()  
8  
9 header = QtGui.QTreeWidgetItem(["Virtual folder tree","Comment"])  
10 #...  
11 folderTree.setHeaderItem(header)    #Another alternative is setHeaderLabels(["Tree","First",...])  
12  
13 root = QtGui.QTreeWidgetItem(folderTree, ["Untagged files"])  
14 root.setData(2, QtCore.Qt.EditRole, 'Some hidden data here')    # Data set to column 2, which is not visible  
15  
16 folder1 = QtGui.QTreeWidgetItem(root, ["Interiors"])  
17 folder1.setData(2, QtCore.Qt.EditRole, 'Some hidden data here')    # Data set to column 2, which is not visible  
18  
19 folder2 = QtGui.QTreeWidgetItem(root, ["Exteriors"])  
20 folder2.setData(2, QtCore.Qt.EditRole, 'Some hidden data here')    # Data set to column 2, which is not visible  
21  
22 folder1_1 = QtGui.QTreeWidgetItem(folder1, ["Bathroom", "Seg was here"])  
23 folder1_1.setData(2, QtCore.Qt.EditRole, 'Some hidden data here')    # Data set to column 2, which is not visible  
24  
25 folder1_2 = QtGui.QTreeWidgetItem(folder1, ["Living room", "Approved by client"])  
26 folder1_2.setData(2, QtCore.Qt.EditRole, 'Some hidden data here')    # Data set to column 2, which is not visible  
27  
28  
29  
30 def printer( treeItem ):  
31     foldername = treeItem.text(0)
```

```

32     comment = treeItem.text(1)
33     data = treeItem.text(2)
34     print foldername + ': ' + comment + ' (' + data + ' )'
35
36
37 folderTree.itemClicked.connect( lambda : printer( folderTree.currentItem() ) )
38
39
40 folderTree.show()
41 sys.exit(app.exec_())

```

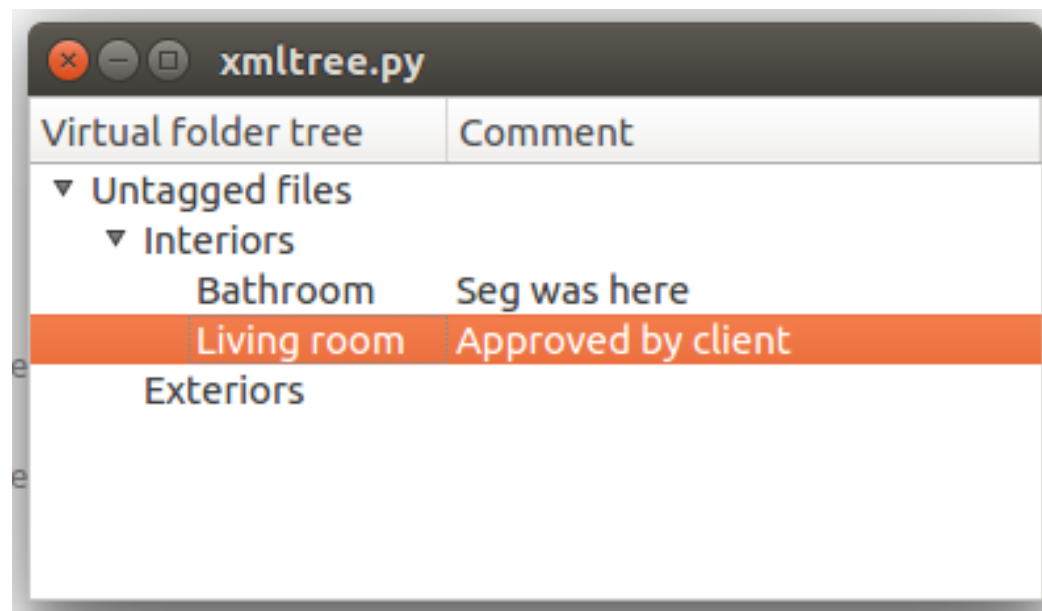


Рис. 16: QTreeWidget

## Работа с XML

```

1 from PyQt4 import QtCore, QtGui, QtXml
2

```

```

3 class XmlHandler(QtXml.QXmlDefaultHandler):
4     def __init__(self, root):
5         QtXml.QXmlDefaultHandler.__init__(self)
6         self._root = root
7         self._item = None
8         self._text = ''
9         self._error = ''
10
11     def startElement(self, namespace, name, qname, attributes):
12         if qname == 'folder' or qname == 'item':
13             if self._item is not None:
14                 self._item = QtGui.QTreeWidgetItem(self._item)
15             else:
16                 self._item = QtGui.QTreeWidgetItem(self._root)
17                 self._item.setData(0, QtCore.Qt.UserRole, qname)
18                 self._item.setText(0, 'Unknown Title')
19                 if qname == 'folder':
20                     self._item.setExpanded(True)
21                 elif qname == 'item':
22                     self._item.setText(1, attributes.value('type'))
23         self._text = ''
24         return True
25
26     def endElement(self, namespace, name, qname):
27         if qname == 'title':
28             if self._item is not None:
29                 self._item.setText(0, self._text)
30         elif qname == 'folder' or qname == 'item':
31             self._item = self._item.parent()
32         return True
33
34     def characters(self, text):
35         self._text += text
36         return True
37
38     def fatalError(self, exception):

```

```

39         print('Parse Error: line %d, column %d:\n %s' % (
40             exception.lineNumber(),
41             exception.columnNumber(),
42             exception.message(),
43             ))
44     return False
45
46     def errorString(self):
47         return self._error
48
49 class Window(QtGui.QTreeWidget):
50     def __init__(self):
51         QtGui.QTreeWidget.__init__(self)
52         self.header().setResizeMode(QtGui.QHeaderView.Stretch)
53         self.setHeaderLabels(['Title', 'Type'])
54         source = QtXml.QXmlInputSource()
55         source.setData(xml)
56         handler = XmlHandler(self)
57         reader = QtXml.QXmlSimpleReader()
58         reader.setContentHandler(handler)
59         reader.setErrorHandler(handler)
60         reader.parse(source)
61
62 xml = """\
63 <root>
64     <folder>
65         <title>Folder One</title>
66         <item type="1">
67             <title>Item One</title>
68         </item>
69         <item type="1">
70             <title>Item Two</title>
71         </item>
72         <item type="2">
73             <title>Item Three</title>
74         </item>

```

```

75     <folder>
76         <title>Folder Two</title>
77         <item type="3">
78             <title>Item Four</title>
79         </item>
80         <item type="0">
81             <title>Item Five</title>
82         </item>
83         <item type="1">
84             <title>Item Six</title>
85         </item>
86     </folder>
87 </folder>
88 <folder>
89     <title>Folder Three</title>
90     <item type="0">
91         <title>Item Six</title>
92     </item>
93     <item type="2">
94         <title>Item Seven</title>
95     </item>
96     <item type="2">
97         <title>Item Eight</title>
98     </item>
99 </folder>
100 </root>
101 """
102
103 if __name__ == '__main__':
104
105     import sys
106     app = QtGui.QApplication(sys.argv)
107     window = Window()
108     window.resize(400, 300)
109     window.show()
110     sys.exit(app.exec_())

```

Title	Type
▼ Folder One	
Item One	1
Item Two	1
Item Three	2
▼ Folder Two	
Item Four	3
Item Five	0
Item Six	1
▼ Folder Three	
Item Six	0
Item Seven	2
Item Eight	2

Рис. 17: Отображение содержимого XML с помощью QTreeWidgetItem

## 7 Задание 5

С использованием библиотеки для разработки веб-приложений CherryPy сделать форму для вывода данных, в табличной форме, хранящихся в формате XML и в базе данных SQLite, с использованием функций из заданий 2 и 3.

# Пример выполнения

Модуль «start.py» содержит класс «start», реализующий стартовую страницу сайта.

```
1 # -*- coding: utf-8 -*-
2 import cherrypy
3 import sys
4 sys.path.insert(0, "./library")
5 from library import library
6 from dataxml import dataxml
7 from bookpage import bookpage
8 from authorpage import authorpage
9 from publpage import publpage
10
11 class start:
12     def __init__(self):
13         self.__lib=library()
14         self.__dataxml=dataxml()
15         self.__dataxml.read('old.xml',self.__lib)
16         self.bookpage=bookpage(self.__lib)
17         self.authorpage=authorpage(self.__lib)
18         self.publpage=publpage(self.__lib)
19     def index(self):
20         return """
21         <a href=bookpage\>книги</a><br>
22         <a href=authorpage\>авторы</a><br>
23         <a href=publpage\>издательства</a><br>
24         """
25     index.exposed=True
26
27 root=start()
28 cherrypy.config.update({
29     'log.screen': True,
30 })
31
32 cherrypy.tree.mount(root)
```



```
33 | if __name__ == '__main__':  
34 |     cherrypy.engine.start()  
35 |     cherrypy.engine.block()
```

Чтобы проверить как работает программа можно пока не описывать полностью классы «bookpage», «authorpage», «publpage». Достаточно создать соответствующие модули и описать в них классы и их конструкторы следующим образом

```
1 | class bookpage:  
2 |     def __init__(self,library):  
3 |         self.__lib=library
```

Запустим программу через консоль

```
1 | > python start.py
```

Затем откроем веб-браузер и в адресной строке введем

```
1 | http://127.0.0.1:8080
```

ИЛИ

```
1 | http://localhost:8080
```

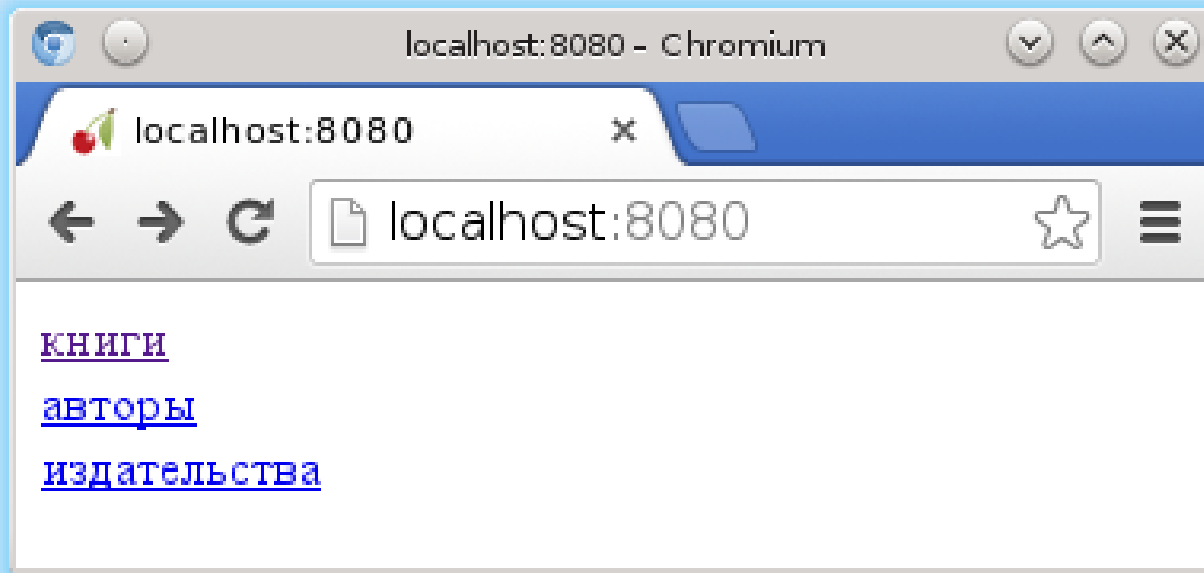


Рис. 18: Стартовая страница сайта

Прервать работу веб-сервера можно нажав в консоли, где была запущена программа комбинацию клавиш «[ctrl]+[c]». Далее полностью опишем классы, реализующие страницы для просмотра и редактирования данных. Модуль «bookpage.py»

```
1 # -*- coding: utf-8 -*-
2
3 class bookpage:
4     def __init__(self, library):
5         self.__lib=library
6
7     def index(self):
```

```

8 s='<a href=..>%s</a>/<a href=addform>%s</a>'%(u'назад',u'добавить')
9 s+='<table><th bgcolor=gray></th><th bgcolor=gray>%s</th><th bgcolor=gray>%s</th><th bgcolor=gray>%s</th><th bgcolor=gray>%s</th>'
10 r=1
11 bg=''
12 for c in self.__lib.getBookCodes():
13     s+='<tr%s><td>%d</td>'%(bg,r)
14     s+='<td>%s</td>'%self.__lib.getBookName(c)
15     s+='<td>%s</td>'%self.__lib.getBookAuthorsBibliostr(c)
16     s+='<td>%s</td>'%self.__lib.getBookPublName(c)
17     s+='<td>%s</td>'%self.__lib.getBookYear(c)
18     s+='<td>%s</td>'%self.__lib.getBookPages(c)
19     s+='<td><a href=editform?code=%s>%s</a></td>'%(c,u'редактировать')
20     s+='<td><a href=delr?code=%s>%s</a></td></tr>'%(c,u'удалить')
21     r+=1
22     if bg: bg=''
23     else: bg=' bgcolor=silver'
24 s+='</table>'
25 return s
26 index.exposed=True
27
28 def publCombo(self,code=0):
29     s='<select name=publ>'
30     for c in self.__lib.getPublCodes():
31         if (code in self.__lib.getBookCodes())and(c==self.__lib.getBookPublCode(code)):v=' selected'
32         else:v=''
33         s+='<option%s value=%s>%s</option>'%(v,str(c),self.__lib.getPublName(c))
34     s+='</select>'
35     return s
36
37 def authorCombo(self,code=0):
38     s='<select name=author>'
39     for c in self.__lib.getAuthorCodes():
40         if not(c in self.__lib.getBookAuthorCodes(code)):
41             s+='<option value=%s>%s</option>'%(str(c),self.__lib.getAuthorBibliostr(c))
42     s+='</select>'
43     return s

```

```

44
45 def authorList(self,code=0):
46     s='<table>'
47     for c in self.__lib.getBookAuthorCodes(code):
48         s+='<tr><td>%s</td><td><a href=delauthor?code=%s&acode=%s>%s</td></tr>'%(self.__lib.getAuthorBibliostr(c),str(code),str(c),u'y
49     s+='</table>'
50     return s
51
52 def bookform(self,code=0,add=True):
53     name,publ,year,pages='',0,0,0
54     if add:a='addaction'
55     else: a='editaction?code=%s'%code
56     if code in self.__lib.getBookCodes():
57         name=self.__lib.getBookName(code)
58         publ=self.__lib.getBookPublCode(code)
59         year=self.__lib.getBookYear(code)
60         pages=self.__lib.getBookPages(code)
61     s='''<form action=%s method=post>
62         <table>
63             <tr><td>%s</td><td><input type=text name=name value='%s'></td></tr>
64             <tr><td>%s</td><td>%s</td></tr>
65             <tr><td>%s</td><td><input type=number name=year value=%s></td></tr>
66             <tr><td>%s</td><td><input type=number name=pages value=%s></td></tr>
67             <tr><td><input type=submit></td><td></td></tr>
68         </table>
69         </form>'''%(a,u'название',name,u'издательство',self.publCombo(publ),u'год',str(year),u'страницы',str(pages))
70     return s
71
72 def addaction(self,name,publ,year,pages):
73     code=self.__lib.getBookNewCode()
74     self.__lib.newBook(code)
75     self.__lib.setBookName(code,name)
76     self.__lib.setBookPubl(code,int(publ))
77     self.__lib.setBookYear(code,year)
78     self.__lib.setBookPages(code,pages)
79     return 'книга добавлена<br><a href=index>назад</a>'

```

```

80 addaction.exposed=True
81
82 def addform(self):
83     s=u'Добавить новую книгу<br>'
84     s+=self.bookform(0)
85     return s
86 addform.exposed=True
87
88 def editform(self,code):
89     s=u'Редактировать книгу<br>'
90     s+=self.bookform(int(code),False)
91     s+='''%s
92         <form action=addauthor?code=%s method=post>
93         <table>
94         <tr><td>%s</td><td><input type=submit value=%s></td>
95         ''%(u'авторы',str(code),self.authorCombo(int(code)),u'добавить')
96     s+=self.authorList(int(code))
97     return s
98 editform.exposed=True
99
100 def editaction(self,code,name,publ,year,pages):
101     self.__lib.setBookName(int(code),name)
102     self.__lib.setBookPubl(int(code),int(publ))
103     self.__lib.setBookYear(int(code),year)
104     self.__lib.setBookPages(int(code),pages)
105     return 'книга изменена<br><a href=index>назад</a>'
106 editaction.exposed=True
107
108 def addauthor(self,code,author):
109     self.__lib.appendBookAuthor(int(code),self.__lib.findAuthorByCode(int(author)))
110     return '%s<br><a href=editform?code=%s>%s</a>'%(u'автор добавлен',str(code),u'назад')
111 addauthor.exposed=True
112
113 def delauthor(self,code,acode):
114     self.__lib.removeBookAuthor(int(code),int(acode))
115     return '%s<br><a href=editform?code=%s>%s</a>'%(u'автор удален',str(code),u'назад')

```

```

116 delauthor.exposed=True
117
118 def delr(self,code):
119     self.__lib.removeBook(int(code))
120     return 'книга удалена<br><a href=index>назад</a>'
121 delr.exposed=True

```

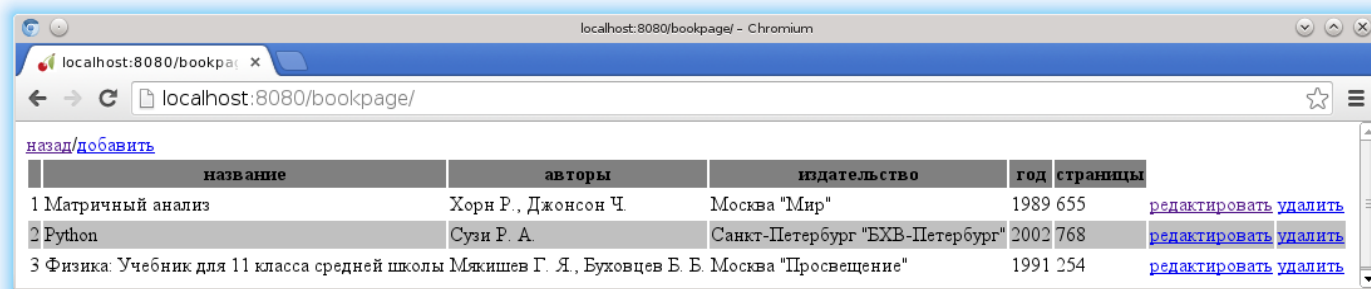


Рис. 19: Стартовая просмотра списка книг

The image shows a web browser window with the title "localhost:8080/bookpage/addform - Chromium". The address bar displays "localhost:8080/bookpage/addform". The page content is in Russian and titled "Добавить новую книгу". It contains a form with the following fields:

- название**: A text input field.
- издательство**: A dropdown menu with "Москва 'Мир'" selected.
- год**: A numeric input field with "0" and up/down arrows.
- страницы**: A numeric input field with "0" and up/down arrows.

At the bottom of the form is a button labeled "Отправить".

Рис. 20: Форма создания новой книги

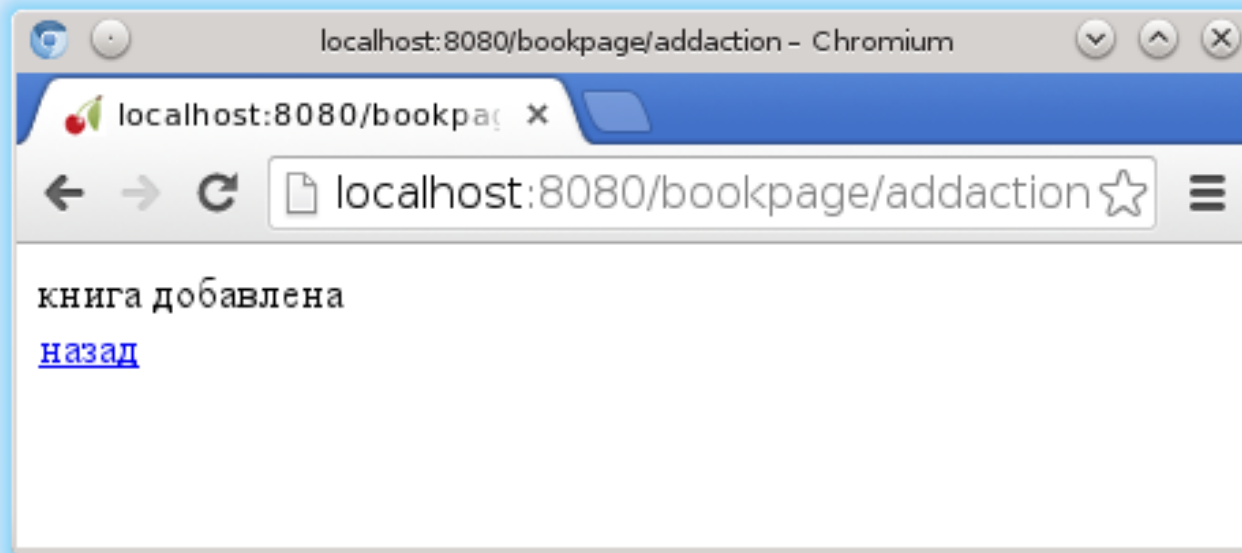


Рис. 21: Результат создания новой книги



localhost:8080/bookpage/editform?code=3 - Chromium

localhost:8080/bookpage x

localhost:8080/bookpage/editform?code=3

### Редактировать книгу

название

издательство

год

страницы

авторы

Мякишев Г. Я. [удалить](#)

Буховцев Б. Б. [удалить](#)

Рис. 22: Форма редактирования книги