ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем

Факультет автоматики и вычислительной техники

Кафедра систем автоматизации управления

**ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ**

|  |  |  |
| --- | --- | --- |
| Ердяков Роман Александрович | | |
| *(Ф.И.О. обучающегося)* | | |
| Информационные системы и технологии, управления технологическими процессами в промышленности | | |
| *(направление подготовки (специальность), направленность (профиль))* | | |
|  | | |
| Место прохождения практики | | ВятГУ |
|  | *(наименование организации, структурного подразделения организации)* | |
|  | | |
| *(наименование организации, структурного подразделения организации)* | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Итоговая оценка: |  | | | | |
| Руководитель  практики от университета |  |  |  |  |  |
|  | *(дата)* |  | *(подпись)* |  | *(Ф.И.О.)* |

Киров, 2025 г.

СОДЕРЖАНИЕ

[1 АНАЛИТИЧЕСКАЯ ЧАСТЬ 3](#_Toc199883936)

[1.1 Сравнительный анализ нескольких технических средств разработки пользовательского интерфейса настольного приложения 3](#_Toc199883937)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 4](#_Toc199883938)

[2.1 Цель работы 4](#_Toc199883939)

[2.2 Инструменты разработки 5](#_Toc199883940)

[2.3 Инструкция по запуску приложения 6](#_Toc199883941)

[2.4 Результаты тестирования программы 7](#_Toc199883942)

[2.4.1 Инициализация приложения тестовыми данными 7](#_Toc199883943)

[2.4.2 Очистка базы данных 7](#_Toc199883944)

[2.4.3 Стартовый экран приложения 7](#_Toc199883945)

[2.4.4 Добавление книги 8](#_Toc199883946)

[2.4.5 Добавление аудиокниги 9](#_Toc199883947)

[2.4.6 Редактирование книги 10](#_Toc199883948)

[2.4.7 Вычисление средней арифметической и медианной стоимости для выбранных элементов списка 11](#_Toc199883949)

[3 ВЫВОД 12](#_Toc199883950)

[ПРИЛОЖЕНИЕ А 13](#_Toc199883951)

[Листинг программы 13](#_Toc199883952)

[Файл app.py 13](#_Toc199883953)

[Файл database.py 14](#_Toc199883954)

[Файл models.py 16](#_Toc199883955)

[Файл view.py 18](#_Toc199883956)

[Файл seeder.py 22](#_Toc199883957)

[ПРИЛОЖЕНИЕ Б 23](#_Toc199883958)

[Библиографический список 23](#_Toc199883959)

# 1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

## 1.1 Сравнительный анализ нескольких технических средств разработки пользовательского интерфейса настольного приложения

В ходе анализа были рассмотрены различные технические средства разработки пользовательского интерфейса настольных приложений. Основное внимание уделялось таким языкам программирования, как C#, Python, Java и Kotlin, а также популярным библиотекам для создания графического интерфейса: Avalonia, Tkinter, wxPython, Swing и JavaFX.

C# совместно с Avalonia или Windows Forms обеспечивает высокую производительность и интеграцию с платформой Windows, однако требует использования среды разработки Visual Studio и в некоторых случаях — лицензий. Java и JavaFX обладают широкой кроссплатформенностью и мощным инструментарием для построения интерфейсов, однако приложения на Java могут потреблять больше ресурсов. Kotlin можно использовать с библиотекой TornadoFX, однако этот стек менее распространён для настольных решений по сравнению с Android-разработкой. wxPython — мощная библиотека для Python с нативным внешним видом интерфейсов, но она сложнее в освоении и требует дополнительных зависимостей. Swing, хоть и давно используется в Java-приложениях, считается морально устаревшим и менее гибким, чем современные аналоги.

В результате сравнительного анализа было выбрано сочетание языка программирования Python и библиотеки Tkinter. Это обусловлено рядом факторов: Tkinter является встроенной библиотекой в стандартную поставку Python, имеет низкий порог входа, хорошо документирован и подходит для быстрого прототипирования. Кроме того, Python обладает широким сообществом, большим количеством библиотек для обработки данных и интеграции с внешними сервисами. Tkinter позволяет разрабатывать кроссплатформенные приложения с базовой графикой, что достаточно для реализации учебных и утилитарных настольных программ.

# 2 ПРАКТИЧЕСКАЯ ЧАСТЬ

## 2.1 Цель работы

Разработка настольного приложения, отображающего коллекцию объектов двух типов — книг и аудиокниг, реализованных в виде классов с различным набором свойств. При этом оба типа сущностей должны наследовать общий базовый класс или реализовывать единый интерфейс, обеспечивающий единообразную работу с коллекцией. Приложение должно позволять визуализировать данные с использованием графического интерфейса.

Реализовать вычисление средней арифметической и медианной стоимости для выбранных элементов списка.

## ****2.2 Инструменты разработки****

Для реализации настольного приложения были выбраны следующие инструменты:

* **Язык программирования:** Python;
* **Библиотека пользовательского интерфейса:** Tkinter;
* **Система управления базами данных:** SQLite.

## 2.3 Инструкция по запуску приложения

1. Для создания схемы базы данных и загрузки тестовых данных вызвать файл seeder.py используя команду: python seeder.py
2. Для запуска приложения вызвать файл app.py используя команду: python app.py

## 2.4 Результаты тестирования программы

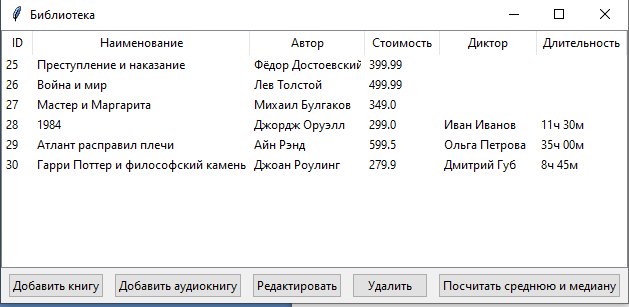
### 2.4.1 Инициализация приложения тестовыми данными



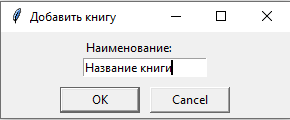
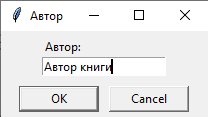
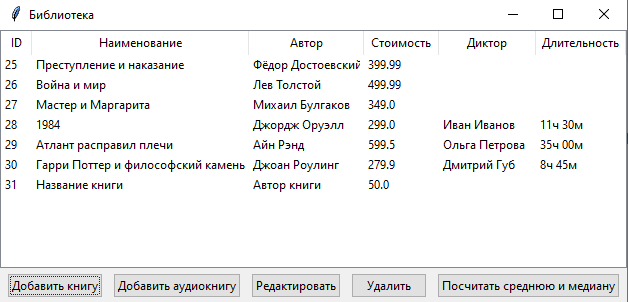
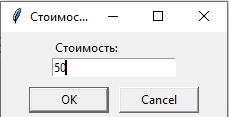
### 2.4.2 Очистка базы данных



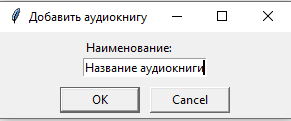
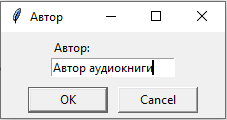
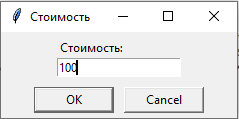
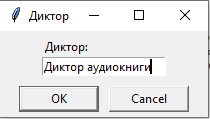
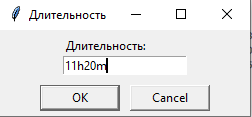
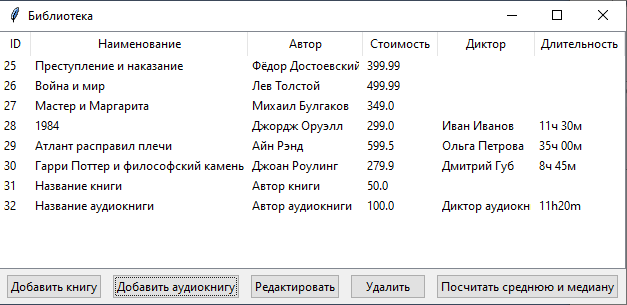
### 2.4.3 Стартовый экран приложения



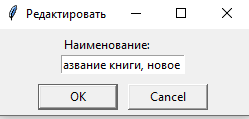
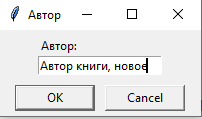
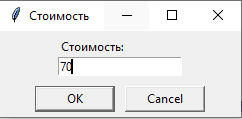
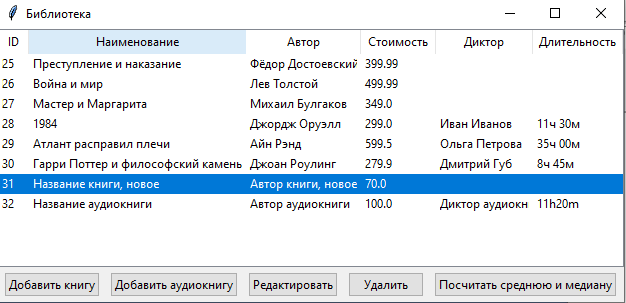
### 2.4.4 Добавление книги

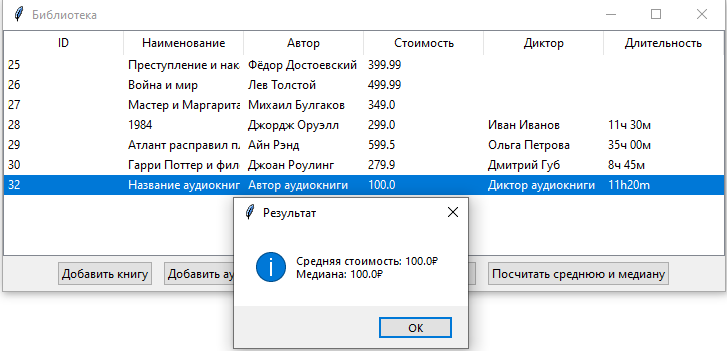
### 2.4.5 Добавление аудиокниги

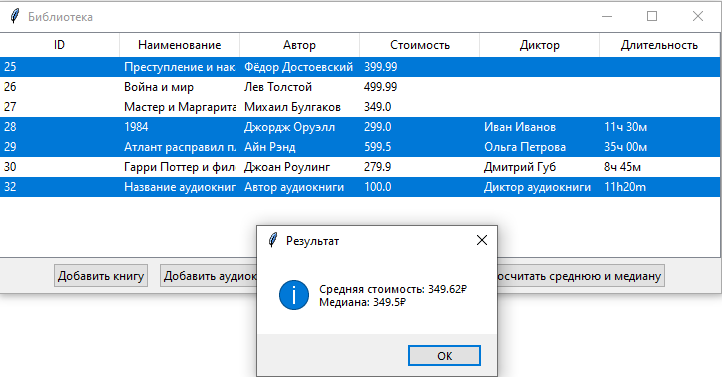
     

### 2.4.6 Редактирование книги

### 2.4.7 Вычисление средней арифметической и медианной стоимости для выбранных элементов списка





# 3 ВЫВОД

В результате проделанной работы было разработано настольное приложение для управления коллекцией книг и аудиокниг с использованием языка программирования Python, библиотеки пользовательского интерфейса Tkinter и базы данных SQLite. Приложение демонстрирует применение объектно-ориентированного подхода с использованием наследования и абстрактных классов, позволяет добавлять, редактировать, удалять и отображать сущности, а также рассчитывать среднюю стоимость и медиану выбранных объектов. Полученный опыт подтвердил удобство и достаточную функциональность выбранных инструментов для реализации настольных приложений среднего уровня сложности.

# ПРИЛОЖЕНИЕ А

## Листинг программы

### Файл app.py

from view import Tinker

if \_\_name\_\_ == "\_\_main\_\_":

tinker = Tinker()

tinker.start()

### Файл database.py

import sqlite3

class Database:

conn = sqlite3.connect('library.db')

conn.row\_factory = sqlite3.Row

@staticmethod

def create\_table():

Database.conn.execute('''

CREATE TABLE IF NOT EXISTS entities (

id INTEGER PRIMARY KEY AUTOINCREMENT,

type TEXT NOT NULL,

name TEXT NOT NULL,

author TEXT NOT NULL,

price REAL NOT NULL,

narrator TEXT,

duration TEXT

)

''')

Database.conn.commit()

@staticmethod

def store(entity):

fields = entity.get\_fields()

fields.pop('id', None)

fields['type'] = entity.type()

column\_names = ", ".join(fields.keys())

placeholders = ", ".join("?" for \_ in fields)

Database.conn.execute(

f'INSERT INTO entities ({column\_names}) VALUES ({placeholders})',

tuple(fields.values())

)

Database.conn.commit()

@staticmethod

def update(entity):

fields = entity.get\_fields()

fields.pop('id', None)

set\_clause = ", ".join(f"{field}=?" for field in fields.keys())

sql = f"UPDATE entities SET {set\_clause} WHERE id=?"

Database.conn.execute(sql, list(fields.values()) + [entity.id])

Database.conn.commit()

@staticmethod

def delete(entity\_id):

Database.conn.execute('DELETE FROM entities WHERE id=?', (entity\_id,))

Database.conn.commit()

@staticmethod

def get\_all():

cursor = Database.conn.execute('SELECT \* FROM entities')

return cursor.fetchall()

@staticmethod

def clear\_db():

Database.conn.execute('DELETE FROM entities')

Database.conn.commit()

### Файл models.py

from database import Database

from abc import ABC, abstractmethod

class Entity(Database, ABC):

registry = {}

def \_\_init\_subclass\_\_(cls, \*\*kwargs):

super().\_\_init\_subclass\_\_(\*\*kwargs)

if hasattr(cls, 'type') and callable(cls.type):

Entity.registry[cls.type()] = cls

def \_\_init\_\_(self, entity\_id, name, author, price):

self.id = entity\_id

self.name = name

self.author = author

self.price = price

@classmethod

@abstractmethod

def from\_row(cls, row):

pass

def get\_fields(self):

return vars(self).copy()

def get\_fields\_values(self):

return list(self.get\_fields().values())

@staticmethod

@abstractmethod

def type():

pass

def store(self):

Database.store(self)

def update(self):

Database.update(self)

@classmethod

def get\_all(cls):

rows = Database.get\_all()

items = []

for row in rows:

type\_ = row["type"]

klass = cls.registry.get(type\_)

if klass:

items.append(klass.from\_row(row))

return items

class Book(Entity):

def \_\_init\_\_(self, entity\_id, name, author, price):

super().\_\_init\_\_(entity\_id, name, author, price)

@staticmethod

def type():

return "book"

@classmethod

def from\_row(cls, row):

id\_, \_, name, author, price, \_, \_ = row

return cls(id\_, name, author, price)

class AudioBook(Entity):

def \_\_init\_\_(self, entity\_id, name, author, price, narrator, duration):

super().\_\_init\_\_(entity\_id, name, author, price)

self.narrator = narrator

self.duration = duration

@staticmethod

def type():

return "audiobook"

@classmethod

def from\_row(cls, row):

id\_, \_, name, author, price, narrator, duration = row

return cls(id\_, name, author, price, narrator, duration)

### Файл view.py

from abc import ABC, abstractmethod

import tkinter as tk

from tkinter import ttk, messagebox, simpledialog

import statistics

from models import Entity, Book, AudioBook

class View(ABC):

@abstractmethod

def start(self):

pass

class Tinker(View):

tinker\_handle = None

def \_\_init\_\_(self):

self.tinker\_handle = tk.Tk()

self.tinker\_handle.title("Библиотека")

self.list\_widget()

self.buttons\_widget()

self.refresh()

def start(self):

self.tinker\_handle.mainloop()

def list\_widget(self):

columns = ('type', 'ID', 'Наименование', 'Автор', 'Стоимость', 'Диктор', 'Длительность')

self.tree = ttk.Treeview(self.tinker\_handle, columns=columns, show='headings', selectmode='extended')

for col in columns:

self.tree.heading(col, text=col)

self.tree.column(col, width=120)

self.tree.pack(fill=tk.BOTH, expand=True)

self.tree.column("type", width=0, stretch=False)

def buttons\_widget(self):

btn\_frame = tk.Frame(self.tinker\_handle)

btn\_frame.pack(pady=5)

ttk.Button(btn\_frame, text="Добавить книгу", command=self.add\_book).grid(row=0, column=0, padx=5)

ttk.Button(btn\_frame, text="Добавить аудиокнигу", command=self.add\_audiobook).grid(row=0, column=1, padx=5)

ttk.Button(btn\_frame, text="Редактировать", command=self.edit\_selected).grid(row=0, column=2, padx=5)

ttk.Button(btn\_frame, text="Удалить", command=self.delete\_selected).grid(row=0, column=3, padx=5)

ttk.Button(btn\_frame, text="Посчитать среднюю и медиану", command=self.calculate\_stats).grid(row=0, column=4, padx=5)

def refresh(self):

for i in self.tree.get\_children():

self.tree.delete(i)

for entity in Entity.get\_all():

values = [entity.type()] + entity.get\_fields\_values()

self.tree.insert('', tk.END, values=values)

def add\_book(self):

name = simpledialog.askstring("Добавить книгу", "Наименование:")

author = simpledialog.askstring("Автор", "Автор:")

price = simpledialog.askfloat("Стоимость", "Стоимость:")

if name and author and price is not None:

Book(None, name, author, price).store()

self.refresh()

def add\_audiobook(self):

name = simpledialog.askstring("Добавить аудиокнигу", "Наименование:")

author = simpledialog.askstring("Автор", "Автор:")

price = simpledialog.askfloat("Стоимость", "Стоимость:")

narrator = simpledialog.askstring("Диктор", "Диктор:")

duration = simpledialog.askstring("Длительность", "Длительность:")

if name and author and price is not None and narrator and duration:

AudioBook(None, name, author, price, narrator, duration).store()

self.refresh()

def edit\_selected(self):

selected = self.tree.selection()

if not selected:

messagebox.showwarning("Предупреждение", "Выберите элемент для редактирования.")

return

item = self.tree.item(selected[0])['values']

type\_ = item[0]

entity\_id = item[1]

name = simpledialog.askstring("Редактировать", "Наименование:", initialvalue=item[2])

author = simpledialog.askstring("Автор", "Автор:", initialvalue=item[3])

price = simpledialog.askfloat("Стоимость", "Стоимость:", initialvalue=item[4])

if type\_ == Book.type():

if name and author and price is not None:

Book(entity\_id, name, author, price).update()

else:

narrator = simpledialog.askstring("Диктор", "Диктор:", initialvalue=item[5])

duration = simpledialog.askstring("Длительность", "Длительность:", initialvalue=item[6])

if name and author and price is not None and narrator and duration:

AudioBook(entity\_id, name, author, price, narrator, duration).update()

self.refresh()

def delete\_selected(self):

selected = self.tree.selection()

if not selected:

messagebox.showwarning("Предупреждение", "Выберите элемент для удаления.")

return

for sel in selected:

item = self.tree.item(sel)

entity\_id = item['values'][1]

Entity.delete(entity\_id)

self.refresh()

def calculate\_stats(self):

selected\_items = self.tree.selection()

if not selected\_items:

messagebox.showinfo("Информация", "Выберите хотя бы один элемент.")

return

prices = []

for item\_id in selected\_items:

item = self.tree.item(item\_id)

values = item['values']

try:

price = float(values[4])

prices.append(price)

except (IndexError, ValueError):

continue

if not prices:

messagebox.showwarning("Ошибка", "Не удалось получить стоимость из выбранных элементов.")

return

avg = round(sum(prices) / len(prices), 2)

med = round(statistics.median(prices), 2)

messagebox.showinfo("Результат", f"Средняя стоимость: {avg}₽\nМедиана: {med}₽")

### Файл seeder.py

import sys

from database import Database

from models import Book, AudioBook

def seed\_data():

Book(None, 'Преступление и наказание', 'Фёдор Достоевский', 399.99).store()

Book(None, 'Война и мир', 'Лев Толстой', 499.99).store()

Book(None, 'Мастер и Маргарита', 'Михаил Булгаков', 349.00).store()

AudioBook(None, '1984', 'Джордж Оруэлл', 299.00, 'Иван Иванов', '11ч 30м').store()

AudioBook(None, 'Атлант расправил плечи', 'Айн Рэнд', 599.50, 'Ольга Петрова', '35ч 00м').store()

AudioBook(None, 'Гарри Поттер и философский камень', 'Джоан Роулинг', 279.90, 'Дмитрий Губ', '8ч 45м').store()

if \_\_name\_\_ == "\_\_main\_\_":

clear\_flag = '--clear' in sys.argv

if clear\_flag:

Database.clear\_db()

print("База очищена")

Database.create\_table()

seed\_data()

print("База наполнена")

# ПРИЛОЖЕНИЕ Б

## Библиографический список

1. Официальная документация Python — <https://docs.python.org/3/>
2. Официальная документация Tkinter — <https://tkdocs.com/>
3. Официальная документация SQLite — <https://sqlite.org/docs.html>