МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

ФГБОУ ВО «ДонНТУ»

Факультет Интеллектуальных систем и программирования

Кафедра «Программная инженерия» им. Л.П. Фельдмана

Лабораторная работа №4

по курсу: «Профессиональная практика программной инженерии»

по теме: «Создание самодокументирующегося кода»

Выполнил:

ст. гр. ПИ-21б

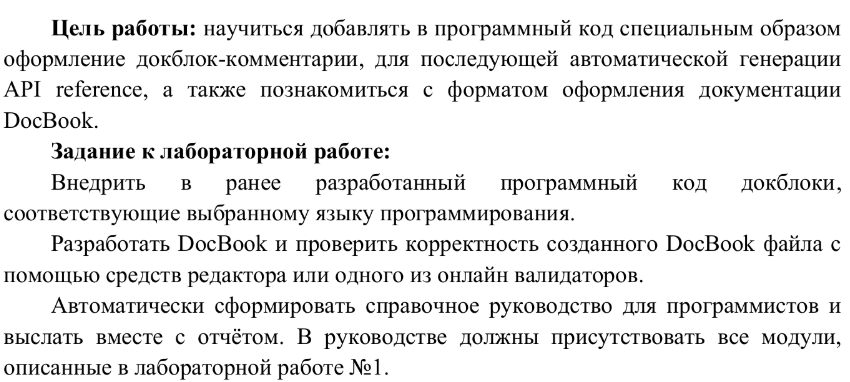
Вавилин М.Е.

Проверил:

асс. каф. ПИ

Ищенко А.П.

ДОНЕЦК – 2025



**Тема:** планировщик задач

**Репозиторий:** https://github.com/the-real-mhlvvln/pppi

**Листинг программы с докблоками:**

class TaskManager:  
 def \_\_init\_\_(self, storage):  
 *"""  
 Инициализация  
 :param storage:  
 """* self.storage = storage  
  
 def add\_task(self, title) -> str:  
 *"""  
 Добавление задачи  
 :param title:  
 :return:  
 """* tasks = self.storage.load\_tasks()  
 task\_id = len(tasks) + 1  
 new\_task = {"id": task\_id, "title": title, "completed": False}  
 self.storage.add\_task(new\_task)  
 return f"Задача '{title}' добавлена."  
  
 def list\_tasks(self):  
 *"""  
 Получить задачи  
 :return:  
 """* tasks = self.storage.load\_tasks()  
 if not tasks:  
 return "Список задач пуст."  
 result = "Список задач:\n"  
 for task in tasks:  
 status = "✓" if task["completed"] else "✗"  
 result += f"{task['id']}. [{status}] {task['title']}\n"  
 return result.strip()  
  
 def complete\_task(self, task\_id):  
 *"""  
 Отметить задачу выполненной  
 :param task\_id:  
 :return:  
 """* tasks = self.storage.load\_tasks()  
 for task in tasks:  
 if task["id"] == task\_id:  
 task["completed"] = True  
 self.storage.save\_tasks(tasks)  
 return f"Задача с ID {task\_id} отмечена как выполненная."  
 return f"Задача с ID {task\_id} не найдена."  
  
 def delete\_task(self, task\_id):  
 *"""  
 Удалить задачу  
 :param task\_id:  
 :return:  
 """* tasks = self.storage.load\_tasks()  
 for task in tasks:  
 if task["id"] == task\_id:  
 tasks.remove(task)  
 self.storage.save\_tasks(tasks)  
 return f"Задача с ID {task\_id} удалена."  
 return f"Задача с ID {task\_id} не найдена."  
  
 def update\_task(self, task\_id, new\_title):  
 *"""  
 Обновить название задачи  
 :param task\_id:  
 :param new\_title:  
 :return:  
 """* tasks = self.storage.load\_tasks()  
 for task in tasks:  
 if task["id"] == task\_id:  
 task["title"] = new\_title  
 self.storage.save\_tasks(tasks)  
 return f"Задача с ID {task\_id} обновлена. Новое название: '{new\_title}'."  
 return f"Задача с ID {task\_id} не найдена."

import json  
import os  
  
class DataStorage:  
 def \_\_init\_\_(self, file\_path="tasks.json"):  
 *"""  
 :param file\_path:  
 """* self.file\_path = file\_path  
  
 def load\_tasks(self):  
 *"""  
 Загружаем таски  
 :return:  
 """* if not os.path.exists(self.file\_path):  
 return []  
 try:  
 with open(self.file\_path, "r", encoding="utf-8") as file:  
 return json.load(file)  
 except json.JSONDecodeError:  
 print("Ошибка: Файл поврежден или содержит некорректные данные.")  
 return []  
  
 def save\_tasks(self, tasks):  
 *"""  
 Сохраняем таски  
 :param tasks:  
 :return:  
 """* try:  
 with open(self.file\_path, "w", encoding="utf-8") as file:  
 json.dump(tasks, file, ensure\_ascii=False, indent=4)  
 except IOError:  
 print("Ошибка: Не удалось сохранить задачи в файл.")  
  
 def add\_task(self, task):  
 *"""Добавление новой задачи."""* tasks = self.load\_tasks()  
 tasks.append(task)  
 self.save\_tasks(tasks)  
  
 def remove\_task(self, task\_id):  
 *"""Удаление задачи по ID."""* tasks = self.load\_tasks()  
 tasks = [task for task in tasks if task.get("id") != task\_id]  
 self.save\_tasks(tasks)  
  
 def update\_task(self, task\_id, updated\_data):  
 *"""Обновление задачи по ID."""* tasks = self.load\_tasks()  
 for task in tasks:  
 if task.get("id") == task\_id:  
 task.update(updated\_data)  
 break  
 self.save\_tasks(tasks)  
  
 def get\_task\_by\_id(self, task\_id):  
 *"""  
 Получить таски по id  
 :param task\_id:  
 :return:  
 """* tasks = self.load\_tasks()  
 for task in tasks:  
 if task.get("id") == task\_id:  
 return task  
 return None

from task\_manager import TaskManager  
from data\_storage import DataStorage  
  
def main():  
 *"""  
 Основная функция  
 :return:  
 """* # Инициализация хранилища и менеджера задач  
 storage = DataStorage("tasks.json")  
 manager = TaskManager(storage)  
  
 while True:  
 print("\n=== Планировщик задач ===")  
 print("1. Добавить задачу")  
 print("2. Просмотреть задачи")  
 print("3. Отметить задачу как выполненную")  
 print("4. Удалить задачу")  
 print("5. Обновить задачу")  
 print("6. Выход")  
  
 choice = input("Выберите действие: ").strip()  
  
 try:  
 if choice == "1":  
 title = input("Введите название задачи: ").strip()  
 if not title:  
 print("Ошибка: Название задачи не может быть пустым.")  
 continue  
 print(manager.add\_task(title))  
  
 elif choice == "2":  
 print(manager.list\_tasks())  
  
 elif choice == "3":  
 task\_id = int(input("Введите ID задачи для отметки: "))  
 print(manager.complete\_task(task\_id))  
  
 elif choice == "4":  
 task\_id = int(input("Введите ID задачи для удаления: "))  
 print(manager.delete\_task(task\_id))  
  
 elif choice == "5":  
 task\_id = int(input("Введите ID задачи для обновления: "))  
 new\_title = input("Введите новое название задачи: ").strip()  
 if not new\_title:  
 print("Ошибка: Новое название задачи не может быть пустым.")  
 continue  
 print(manager.update\_task(task\_id, new\_title))  
  
 elif choice == "6":  
 print("До свидания!")  
 break  
  
 else:  
 print("Неверный выбор. Попробуйте снова.")  
  
 except ValueError:  
 print("Ошибка: Введено некорректное значение. Пожалуйста, введите число.")  
 except Exception as e:  
 print(f"Произошла ошибка: {e}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Документация DocBook:**

<?xml version='1.0'?>

<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN"

"http://www.oasis-open.org/docbook/xml/4.4/docbookx.dtd">

<book>

<title>Справочное руководство по планировщику задач</title>

<!-- Глава 1: Введение -->

<chapter id="introduction">

<title>Введение</title>

<para>

Это справочное руководство описывает функциональность планировщика задач.

Проект разделен на несколько модулей, каждый из которых выполняет свою роль:

управление задачами, хранение данных и пользовательский интерфейс.

</para>

</chapter>

<!-- Глава 2: Модули -->

<chapter id="modules">

<title>Модули</title>

<!-- Раздел: TaskManager -->

<section id="task-manager">

<title>Модуль управления задачами (TaskManager)</title>

<para>

Этот модуль отвечает за управление задачами, включая их создание,

обновление, удаление и отметку как выполненных.

</para>

<programlisting>

<![CDATA[

class TaskManager:

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

"""

Инициализация TaskManager с использованием DataStorage.

:param storage: Экземпляр класса DataStorage для хранения задач.

"""

self.storage = storage

def add\_task(self, title):

"""

Добавление новой задачи.

:param title: Название задачи.

:type title: str

:return: Сообщение об успешном добавлении задачи.

:rtype: str

"""

tasks = self.storage.load\_tasks()

task\_id = len(tasks) + 1

new\_task = {"id": task\_id, "title": title, "completed": False}

self.storage.add\_task(new\_task)

return f"Задача '{title}' добавлена."

def list\_tasks(self):

"""

Просмотр всех задач.

:return: Список задач в удобочитаемом формате.

:rtype: str

"""

tasks = self.storage.load\_tasks()

if not tasks:

return "Список задач пуст."

result = "Список задач:\n"

for task in tasks:

status = "✓" if task["completed"] else "✗"

result += f"{task['id']}. [{status}] {task['title']}\n"

return result.strip()

def complete\_task(self, task\_id):

"""

Отметка задачи как выполненной.

:param task\_id: Уникальный идентификатор задачи.

:type task\_id: int

:return: Сообщение об успешной отметке задачи.

:rtype: str

"""

tasks = self.storage.load\_tasks()

for task in tasks:

if task["id"] == task\_id:

task["completed"] = True

self.storage.save\_tasks(tasks)

return f"Задача с ID {task\_id} отмечена как выполненная."

return f"Задача с ID {task\_id} не найдена."

def delete\_task(self, task\_id):

"""

Удаление задачи.

:param task\_id: Уникальный идентификатор задачи.

:type task\_id: int

:return: Сообщение об успешном удалении задачи.

:rtype: str

"""

tasks = self.storage.load\_tasks()

for task in tasks:

if task["id"] == task\_id:

tasks.remove(task)

self.storage.save\_tasks(tasks)

return f"Задача с ID {task\_id} удалена."

return f"Задача с ID {task\_id} не найдена."

def update\_task(self, task\_id, new\_title):

"""

Обновление названия задачи.

:param task\_id: Уникальный идентификатор задачи.

:type task\_id: int

:param new\_title: Новое название задачи.

:type new\_title: str

:return: Сообщение об успешном обновлении задачи.

:rtype: str

"""

tasks = self.storage.load\_tasks()

for task in tasks:

if task["id"] == task\_id:

task["title"] = new\_title

self.storage.save\_tasks(tasks)

return f"Задача с ID {task\_id} обновлена. Новое название: '{new\_title}'."

return f"Задача с ID {task\_id} не найдена."

]]>

</programlisting>

</section>

<!-- Раздел: DataStorage -->

<section id="data-storage">

<title>Модуль хранения данных (DataStorage)</title>

<para>

Этот модуль отвечает за сохранение и загрузку задач из файла JSON.

Он предоставляет методы для работы с данными: загрузка, сохранение,

добавление, удаление и обновление задач.

</para>

<programlisting>

<![CDATA[

import json

import os

class DataStorage:

def \_\_init\_\_(self, file\_path="tasks.json"):

"""

Инициализация DataStorage.

:param file\_path: Путь к файлу для хранения задач.

:type file\_path: str

"""

self.file\_path = file\_path

def load\_tasks(self):

"""

Загрузка задач из файла.

:return: Список задач.

:rtype: list

"""

if not os.path.exists(self.file\_path):

return []

try:

with open(self.file\_path, "r", encoding="utf-8") as file:

return json.load(file)

except json.JSONDecodeError:

print("Ошибка: Файл поврежден или содержит некорректные данные.")

return []

def save\_tasks(self, tasks):

"""

Сохранение задач в файл.

:param tasks: Список задач для сохранения.

:type tasks: list

"""

try:

with open(self.file\_path, "w", encoding="utf-8") as file:

json.dump(tasks, file, ensure\_ascii=False, indent=4)

except IOError:

print("Ошибка: Не удалось сохранить задачи в файл.")

def add\_task(self, task):

"""

Добавление новой задачи.

:param task: Словарь с данными задачи.

:type task: dict

"""

tasks = self.load\_tasks()

tasks.append(task)

self.save\_tasks(tasks)

def remove\_task(self, task\_id):

"""

Удаление задачи по ID.

:param task\_id: Уникальный идентификатор задачи.

:type task\_id: int

"""

tasks = self.load\_tasks()

tasks = [task for task in tasks if task.get("id") != task\_id]

self.save\_tasks(tasks)

def update\_task(self, task\_id, updated\_data):

"""

Обновление задачи по ID.

:param task\_id: Уникальный идентификатор задачи.

:type task\_id: int

:param updated\_data: Новые данные для задачи.

:type updated\_data: dict

"""

tasks = self.load\_tasks()

for task in tasks:

if task.get("id") == task\_id:

task.update(updated\_data)

break

self.save\_tasks(tasks)

def get\_task\_by\_id(self, task\_id):

"""

Получение задачи по ID.

:param task\_id: Уникальный идентификатор задачи.

:type task\_id: int

:return: Задача или None, если задача не найдена.

:rtype: dict or None

"""

tasks = self.load\_tasks()

for task in tasks:

if task.get("id") == task\_id:

return task

return None

]]>

</programlisting>

</section>

<!-- Раздел: UI -->

<section id="user-interface">

<title>Модуль пользовательского интерфейса (UI)</title>

<para>

Этот модуль предоставляет текстовый интерфейс для взаимодействия с

пользователями. Он позволяет добавлять, удалять, просматривать и

обновлять задачи через консоль.

</para>

<programlisting>

<![CDATA[

from task\_manager import TaskManager

from data\_storage import DataStorage

def main():

"""

Точка входа в программу. Предоставляет текстовый интерфейс для управления задачами.

"""

storage = DataStorage("tasks.json")

manager = TaskManager(storage)

while True:

print("\n=== Планировщик задач ===")

print("1. Добавить задачу")

print("2. Просмотреть задачи")

print("3. Отметить задачу как выполненную")

print("4. Удалить задачу")

print("5. Обновить задачу")

print("6. Выход")

choice = input("Выберите действие: ").strip()

try:

if choice == "1":

title = input("Введите название задачи: ").strip()

if not title:

print("Ошибка: Название задачи не может быть пустым.")

continue

print(manager.add\_task(title))

elif choice == "2":

print(manager.list\_tasks())

elif choice == "3":

task\_id = int(input("Введите ID задачи для отметки: "))

print(manager.complete\_task(task\_id))

elif choice == "4":

task\_id = int(input("Введите ID задачи для удаления: "))

print(manager.delete\_task(task\_id))

elif choice == "5":

task\_id = int(input("Введите ID задачи для обновления: "))

new\_title = input("Введите новое название задачи: ").strip()

if not new\_title:

print("Ошибка: Новое название задачи не может быть пустым.")

continue

print(manager.update\_task(task\_id, new\_title))

elif choice == "6":

print("До свидания!")

break

else:

print("Неверный выбор. Попробуйте снова.")

except ValueError:

print("Ошибка: Введено некорректное значение. Пожалуйста, введите число.")

except Exception as e:

print(f"Произошла ошибка: {e}")

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

main()

]]>

</programlisting>

</section>

</chapter>

<!-- Глава 3: API Reference -->

<chapter id="api-reference">

<title>API Reference</title>

<para>

Полный список доступных методов и их описания можно найти в соответствующих

модулях выше. Для быстрого доступа используйте ссылки:

</para>

<itemizedlist>

<listitem><para><link linkend="task-manager">TaskManager</link></para></listitem>

<listitem><para><link linkend="data-storage">DataStorage</link></para></listitem>

<listitem><para><link linkend="user-interface">UI</link></para></listitem>

</itemizedlist>

</chapter>

</book>

**DOCX-версия документации, сгенерированная на сайте** [**https://pandoc.org/try/**](https://pandoc.org/try/)

