МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

Пояснительная записка

К лабораторной работе №2

по дисциплине

Технологии разработки программного обеспечения

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Чернобаев И.Д.

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Савкин А.Е.

(подпись) (фамилия, и.,о.)

М23-ИВТ-1

(шифр группы)

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Задание к выполнению лабораторной работы №2**

**Трекер клавиатуры**

# Текст задания

* Реализовать трекер событий клавиатуры (какая клавиша, какой событие с клавишей произошло).
* Трекер запустить в отдельном потоке.
* Реализовать паттерн "Наблюдатель + Итератор":
* Каждое событие трекер передает всем подписанным "слушателям" (on\_next).
* Подписчики должны отобразить полученную информацию в файл.
* Реализовать завершение работы трекера по вводу определенной комбинации клавиш (on\_complete).
* Реализовать обработку ошибок (on\_error).

# Теоретическая часть:

**1. Модуль pynput в Python, управление клавиатурой и мышью.**

Модуль pynput позволяет перехватывать и имитировать события устройств ввода, а именно клавиатуру и мышь. Модуль содержит подмодули для каждого типа поддерживаемого устройства ввода: pynput.mouse и pynput.keyboard

**2. Наблюдатель** — это поведенческий паттерн, который позволяет объектам оповещать другие объекты об изменениях своего состояния.

При этом наблюдатели могут свободно подписываться и отписываться от этих оповещений.

Сложность: \*\*

Популярность: \*\*\*

Применимость: Наблюдатель можно часто встретить в Python коде, особенно там, где применяется событийная модель отношений между компонентами. Наблюдатель позволяет отдельным компонентам реагировать на события, происходящие в других компонентах.

Признаки применения паттерна: Наблюдатель можно определить по механизму подписки и методам оповещения, которые вызывают компоненты программы.

**3. Итератор** — это поведенческий паттерн, позволяющий последовательно обходить сложную коллекцию, без раскрытия деталей её реализации.

Благодаря Итератору, клиент может обходить разные коллекции одним и тем же способом, используя единый интерфейс итераторов.

Сложность: \*\*

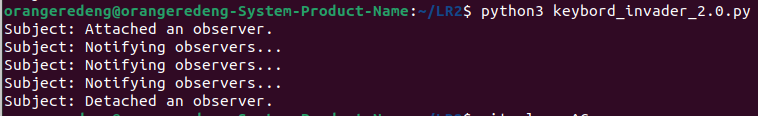
Популярность:\*\*\*

**Применимость:** Паттерн можно часто встретить в Python-коде, особенно в программах, работающих с разными типами коллекций, и где требуется обход разных сущностей.

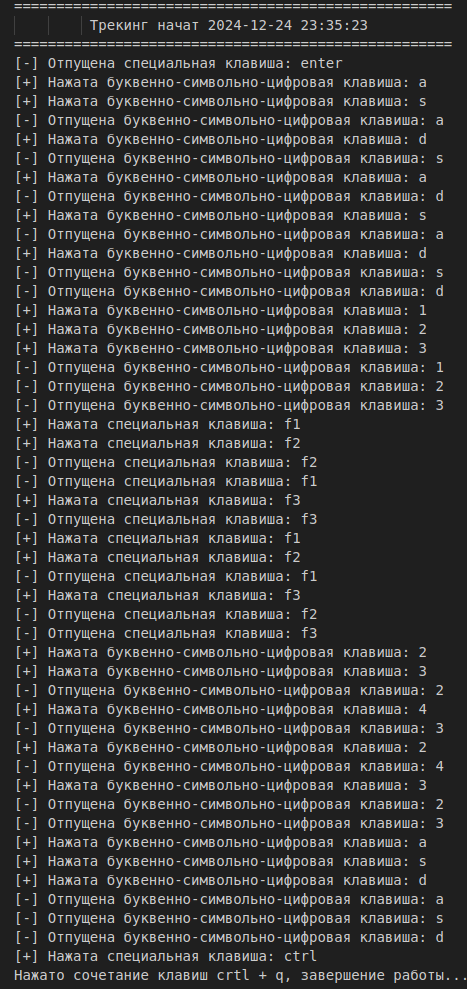
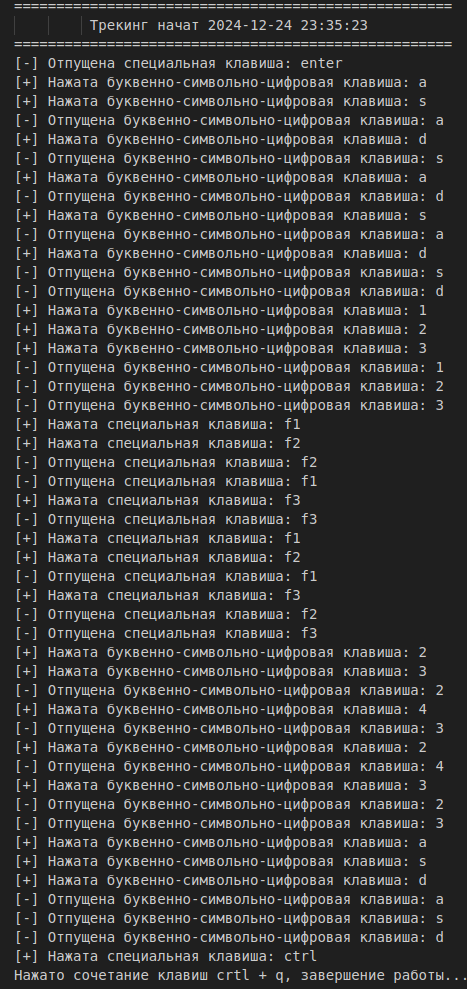
**Признаки применения паттерна:** Итератор легко определить по методам навигации (например, получения следующего/предыдущего элемента и т.д.). Код использующий итератор зачастую вообще не имеет ссылок на коллекцию, с которой работает итератор. Итератор либо принимает коллекцию в параметрах конструктора при создании, либо возвращается самой коллекцией.

# Ход работы

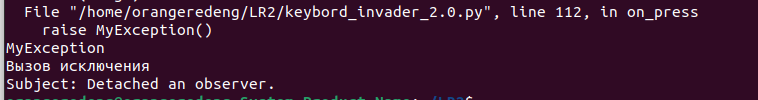
Работа программы с отображением работы паттерна Наблюдателя (паттерн итератор реализован на уровне библиотеки pynput):

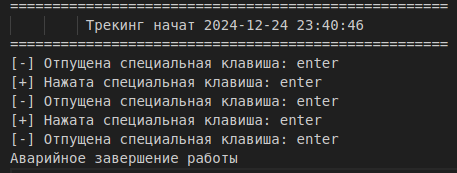


Визуализация стандартного вывода в файл (завершение работы по нажатию сочетания клавиш):



Вызов исключения при имитации ошибки и аварийное завершение работы:





# Вывод

В ходе выполнения лабораторной работы были освоен инструмент PyNput для логгирования событий клавиатуры, освоены паттерны наблюдатель и итератор.

# Код

**from** abc **import** ABC, abstractmethod

**from** typing **import** List

**import** datetime

**from** pynput **import** keyboard

time\_format = "%Y-%m-%d %H:%M:%S"

beauty = "====================================================\n"

"""

Немного глобально объявленных переменных для красивостей.

"""

**class** **MyException**(**Exception**): **pass**

"""

Класс для собственного исключения.

"""

**class** **Subject**(**ABC**):

"""

Интерфейс издателя объявляет набор методов для управлениями подписчиками.

"""

@abstractmethod

**def** **attach**(self, observer: Observer) -> None:

"""

Присоединяет наблюдателя к издателю.

"""

**pass**

@abstractmethod

**def** **detach**(self, observer: Observer) -> None:

"""

Отсоединяет наблюдателя от издателя.

"""

**pass**

@abstractmethod

**def** **notify**(self) -> None:

"""

Уведомляет всех наблюдателей о событии.

"""

**pass**

**class** **KeyCapture**(**Subject**):

"""

Издатель владеет некоторым важным состоянием и оповещает наблюдателей о его

изменениях.

"""

\_state: str = None

"""

Для удобства в этой переменной хранится состояние Издателя, необходимое всем

подписчикам.

"""

\_observers: List[Observer] = []

"""

Список подписчиков. В реальной жизни список подписчиков может храниться в

более подробном виде (классифицируется по типу события и т.д.)

"""

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

self.tracker = keyboard.Listener(

on\_press=self.on\_press,

on\_release=self.on\_release)

self.hotkey\_tracker = keyboard.GlobalHotKeys({

'<ctrl>+q': self.on\_completed})

"""

Функция инициализации трекера клавиш и трекера сочетания клавиш для заверщения программы.

"""

**def** **start\_tracking**(self):

self.tracker.start()

self.hotkey\_tracker.start()

**try**:

self.tracker.join()

self.hotkey\_tracker.join()

**except** MyException **as** e:

print(f'Вызов исключения')

"""

Функция запуска трекеров в отедельных потоках.

"""

**def** **on\_completed**(self):

self.\_state = 'Нажато сочетание клавиш crtl + q, завершение работы...\n'

self.tracker.stop()

self.hotkey\_tracker.stop()

self.notify()

"""

Функция обработки завершения работы.

"""

**def** **on\_press**(self, key):

**try**:

self.\_state = f'[+] Нажата буквенно-символьно-цифровая клавиша: {key.char}\n'

**except** AttributeError:

self.\_state = f'[+] Нажата специальная клавиша: {key.name}\n'

**if** key == keyboard.Key.esc:

self.\_state = f'Аварийное завершение работы...\n'

self.notify()

**raise** MyException()

self.notify()

"""

Функция обработки нажатий вкючая генерацию исключения при нажатии Esc и уведомление подписчиков о событии.

"""

**def** **on\_release**(self, key):

**try**:

self.\_state = f'[-] Отпущена буквенно-символьно-цифровая клавиша: {key.char}\n'

**except** AttributeError:

self.\_state = f'[-] Отпущена специальная клавиша: {key.name}\n'

self.notify()

"""

Функция обработки "отжатий" и уведомление подписчиков о событии.

"""

**def** **notify**(self) -> None:

print("Subject: Notifying observers...")

**for** observer **in** self.\_observers:

observer.update(self)

"""

Запуск обновления в каждом подписчике.

"""

**def** **attach**(self, observer: Observer) -> None:

print("Subject: Attached an observer.")

self.\_observers.append(observer)

**def** **detach**(self, observer: Observer) -> None:

print("Subject: Detached an observer.")

self.\_observers.remove(observer)

"""

Методы управления подпиской.

"""

**class** **Observer**(**ABC**):

"""

Интерфейс Наблюдателя объявляет метод уведомления, который издатели

используют для оповещения своих подписчиков.

"""

@abstractmethod

**def** **update**(self, subject: Subject) -> None:

"""

Получить обновление от субъекта.

"""

**pass**

"""

Конкретные Наблюдатели реагируют на обновления, выпущенные Издателем, к которому

они прикреплены.

"""

**class** **FileWriter**(**Observer**):

**def** **\_\_init\_\_**(self, filename):

self.filename = filename

**with** open(self.filename, 'a') **as** f:

now = datetime.datetime.now(datetime.timezone.utc).astimezone()

f.write(beauty)

f.write(f"Трекинг начат {now:{time\_format}}".center(len(beauty)-1) + '\n')

f.write(beauty)

"""

Инициализация наблюдателя, записывающего в файл события.

"""

**def** **update**(self, subject: Subject) -> None:

**with** open(self.filename, 'a') **as** f:

f.write(subject.\_state)

"""

Получить обновление от субъекта.

"""

**def** **main**():

# Создание Издателя

subject = KeyCapture()

# Создание Наблюдателя

observer = FileWriter("keylog.txt")

# Присоединяем наблюдателя к издателю

subject.attach(observer)

# Начинаем логгирование клавиатурных событий

subject.start\_tracking()

# Отсоединяем наблюдателя от издателя по завершению логгирования

subject.detach(observer)

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

main()