ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ УФИМСКИЙ УНИВЕРСИТЕТ НАУКИ И ТЕХНОЛОГИЙ

Кафедра ВМиК

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3**

по предмету **«Объектно-ориентированное программирование»**

Выполнил: студент группы МО-203б

Ярошко Е. В.

Проверил:  
доцент каф. ВМиК

Котельников В. А.

**Уфа 2025 г.**

**Цель лабораторной работы**

Создать GUI-приложение с определённой логикой работы программы. Использовать схему MVC для создания приложения.

**Задание**

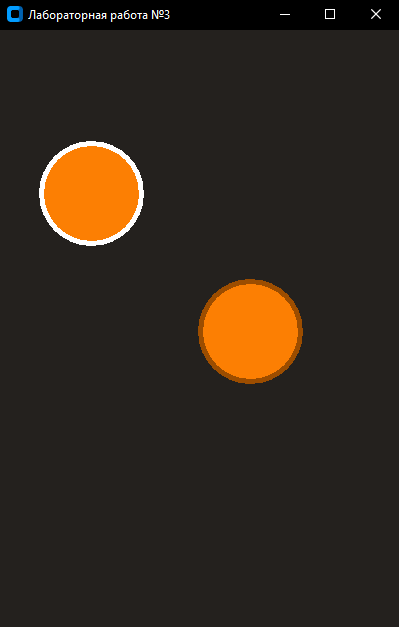
1. Создать простейшее приложение с GUI, содержащее:определение простейшего класса CCircle с координатами и постоянным радиусом; определение простого собственного объекта-контейнера для хранения коллекции создаваемых объектов; форму с объектом для рисования (например, PaintBox). Реализовать следующее поведение: при нажатии мышкой на форме создается новый объект CCircle с координатами нажатия и помещается в контейнер; при событии Paint должны отрисовываться на форме все объекты из контейнера; форма должна позволять изменять свой размер и обрабатывать это изменение. Реализовать следующее поведение с выделением объектов: выделенными могут быть несколько объектов; логика поведения, в какой момент объект становится выделенным или перестаёт быть выделенным, должна соответствовать хотя бы одной популярной программе, где есть манипуляции графическими объектами (например, Microsoft Word или Visio); выделение происходит при нажатии ЛКМ (левой клавиши мыши) на объект CCircle на форме; при нажатии на область формы, где пересекаются несколько кругов, могут выделяться они все или какой-то один из них (преподаватель может потребовать быстро изменить эту логику); при нажатии на кнопку Del, все выделенные объекты должны удаляться; при выделении объекта с помощью ЛКМ и удерживаемой клавиши Ctrl, выделенными становятся несколько объектов.
2. Создать простейшее приложение с GUI, содержащее: три целых числа A, B и C со значением в пределах от 0 до 100; каждое из чисел должно отображаться и редактироваться в 3 разных компонентах: в textBox, numericUpDown, trackBar (или аналогичных в других языках), при этом редактирование числа в одном поле должно приводить к изменению отображения этого числа во всех других полях; второе число всегда должно быть не меньше первого и не больше третьего; приложение должно сохранять значения чисел между запусками (запоминать при закрытии и восстанавливать значения при открытии). Разработанное приложение должно быть реализовано в стиле MVC: хранение трёх чисел должно быть организовано в виде отдельного объекта-модели; все пересчёты, проверки и сохранение должны выполняться в объекте-модели; изменение A и C должно реализовывать разрешающее поведение (при нарушении ограничений порядка модель сама перестраивается, чтобы их выполнить, введённое пользователем значение A и C сохраняется); изменение B должно реализовывать запрещающее поведение (при нарушении ограничений порядка модель откатывает внесённые пользователем изменения) или ограничивающее поведение (при нарушении ограничений порядка модель корректирует введённое значение так, чтобы оно было как можно ближе к заданному пользователем).

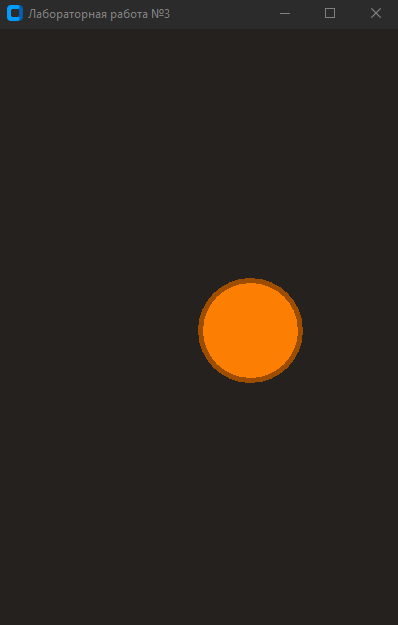
**Ход выполнения лабораторной работы**

Для выполнения данной лабораторной работы использовалась среда разработки, включающая в себя ядро языка Python 3.13.2 с подключённой библиотекой Customtkinter. Редактирование кода выполнялось в программе Visual Studio Code, в качестве компилятора использовался нативный компилятор Python.

При выполнении первого задания был реализован главный класс CCircle, содержащий в себе атрибуты для отрисовки круга на форме, а также координаты положения на форме. В данном классе реализованы методы графической отрисовки самого объекта, определения положения мыши относительно объекта, а также методы выделения и уничтожения. Также был реализован класс Container, который добавляет все объекты класса CCircle в расширяемый список. С помощью класса Container происходит вызов методов при нажатии кнопкой мышки, который, в свою очередь, вызывает методы на проверку и выделение у объектов CCircle. Также в Container реализовано событие Paint, которое перехватывает все вызовы методов у классов и инициирует перерисовку всех фигур в соответствии с изменениями, вызванными данными методами. В главном классе приложения App настраивается окно приложения, а также создаётся главная форма для отображения кругов (Canvas, яв. элементом библиотеки Tkinter, которую включает в себя Customtkinter), после чего происходит привязка кнопок периферийных устройств пользователя к методам объекта Container.

Реализован весь необходимый функционал: создание кругов на форме и в контейнере, выделение кругов (обводка белым цветом), удаление кругов при нажатии Delete (Рисунок 1, 2):

Рисунок 1 — Создание и выделение кругов

Рисунок 2 — Удаление выделенного круга

При выполнении второго задания лабораторной работы использовалась та же среда разработки, что и в первом задании.

В ходе выполнения был создан класс Model, отвечающий за логику приложения и корректировку трёх значений: A, B и C. Также был создан класс Controller, который отвечает за отрисовку элементов интерфейса и их привязку к методам класса. Т. к. по заданию лабораторной работы необходимо, чтобы данные элементы обновлялись в соответствии с решениями Model, то модель View реализована в классе Controller. В главном классе программы App настраивается главное окно.

Т. к. в библиотеке Customtkinter не реализован элемент Spinbox, то он реализован в отдельном файле spinbox.py, который импортируется в программу.

При компиляции программы высвечивается главное окно приложения (Рисунок 3), позволяющее задавать значения A, B и C. При нарушении правил (A <= B < = C) модель сама корректирует значения и не позволяет пользователю нарушить их.

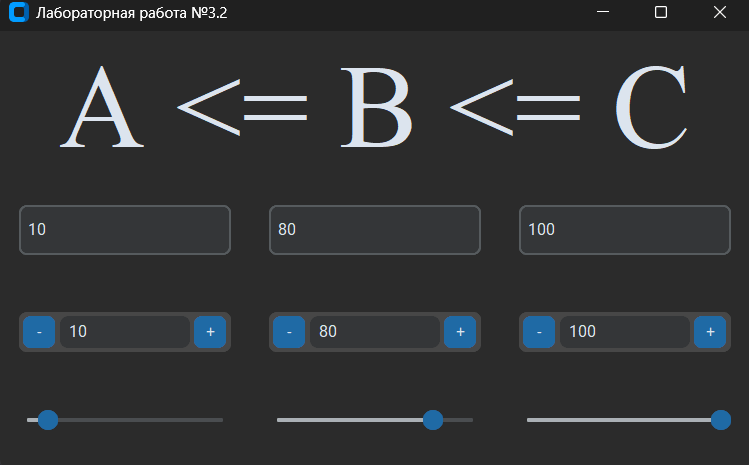


Рисунок 3 — Главное окно приложения

**Выводы по лабораторной работе**

Создали GUI-приложение с определённой логикой работы программы. Использовали схему MVC для создания приложения.

**Приложение 1**

Исходные коды программ по заданиям

**Задание 1**

Файл lab3.1.py: <https://github.com/qvap/OOP--Lab-3.1/blob/main/lab3.1.py>

Исходный код:

import customtkinter as ctk

from tkinter import Canvas

from math import sqrt

class CCircle(): # Базовый класс круга

def \_\_init\_\_(self, canvas, x, y):

# Private

self.\_\_radius = 50

self.\_\_color = "#fc7f03"

self.\_\_border\_color = "#9c4d00"

self.\_\_selected\_border\_color = "#FFFFFF"

self.\_\_draw\_border = False

self.\_\_x = x

self.\_\_y = y

# Public

self.canvas = canvas

print(f"New point: {x}, {y}")

def draw(self): # Отрисовывает круг на Canvas

\_\_chosen\_border\_color = self.\_\_selected\_border\_color if self.\_\_draw\_border else self.\_\_border\_color

self.canvas.create\_oval(self.\_\_x - self.\_\_radius, self.\_\_y - self.\_\_radius,\

self.\_\_x + self.\_\_radius, self.\_\_y + self.\_\_radius, fill = self.\_\_color, width = 5, outline = \_\_chosen\_border\_color)

def set\_border(self, should\_draw: bool):

self.\_\_draw\_border = should\_draw

def mousecheck(self, x: int, y: int) -> bool: # Проверяет, наход. ли точка внутри круга

return ((x - self.\_\_x)\*\*2 + (y - self.\_\_y)\*\*2) <= self.\_\_radius\*\*2

class Container():

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

# Private

self.\_\_container = list()

self.\_\_selected\_container = list()

self.\_\_multiple\_selection = False

# Public

self.canvas = canvas

def container\_append(self, object: CCircle): # Добавляет в контейнер новый круг

print(f"Appending new object: {object}")

self.\_\_container.append(object)

def create\_object(self, point): # Создаёт новый круг и добавляет в список

self.deselect\_objects()

self.container\_append(CCircle(x=point.x, y=point.y, canvas=self.canvas))

def select\_objects(self, point): # Выделяет круги (в зависимости от \_\_multiple\_selection меняется поведение)

self.deselect\_objects()

self.\_\_selected\_container.extend(list(filter(lambda x: x.mousecheck(point.x, point.y), self.\_\_container)))

for circle in set(self.\_\_selected\_container):

circle.set\_border(True)

if not (self.\_\_multiple\_selection):

break

def deselect\_objects(self, \*args): # Снимает выделение со всех кругов

if not (self.\_\_multiple\_selection):

for circle in self.\_\_selected\_container:

circle.set\_border(False)

self.\_\_selected\_container.clear()

def delete\_objects(self, \*args): # Удаляет выделенные объекты

for obj in self.\_\_selected\_container:

self.\_\_container.remove(obj)

del obj

self.\_\_selected\_container.clear()

def initiate\_selection(self, \*args): # Включает множественное выделение

self.\_\_multiple\_selection = True

def stop\_selection(self, \*args): # Выключает множественное выделение

self.\_\_multiple\_selection = False

def \_\_getattribute\_\_(self, name): # событие Paint

attr = super().\_\_getattribute\_\_(name)

if callable(attr):

def wrapper(\*args, \*\*kwargs):

result = attr(\*args, \*\*kwargs)

self.canvas.delete("all")

for circle in self.\_\_container:

circle.draw()

return result

return wrapper

return attr

class App(ctk.CTk):

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

super().\_\_init\_\_()

self.title("Лабораторная работа №3")

self.geometry("400x600")

self.rowconfigure(0, weight=1)

self.columnconfigure(0, weight=1)

self.canvas = Canvas(master=self, bg="#24211e", highlightbackground="#24211e")

self.canvas.grid(row=0, column=0, sticky="nsew")

self.container = Container(canvas=self.canvas)

self.bind("<Button-1>", self.container.create\_object)

self.bind("<Button-3>", self.container.select\_objects)

self.bind\_all("<Escape>", self.container.deselect\_objects)

self.bind\_all("<Delete>", self.container.delete\_objects)

self.bind("<KeyPress-Control\_L>", self.container.initiate\_selection)

self.bind("<KeyRelease-Control\_L>", self.container.stop\_selection)

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

app = App()

app.mainloop()

**Задание 2**

Файл lab3.2.py: <https://github.com/qvap/OOP--Lab-3.2/blob/main/lab3.2.py>

Исходный код:

import customtkinter as ctk

from tkinter import END

import spinbox as sb

import json

import os

class Model():

def \_\_init\_\_(self, show\_numbers):

self.\_\_low\_limit = 0

self.\_\_up\_limit = 100

self.show\_numbers = show\_numbers

self.settings\_path = "settings.json"

if os.path.exists(self.settings\_path):

with open(self.settings\_path, "r") as f:

self.\_\_a, self.\_\_b, self.\_\_c = json.load(f)

else:

self.\_\_a, self.\_\_b, self.\_\_c = 0, 50, 100

def reset\_numbers(self): # Вставляет значения из модели в виджеты

self.show\_numbers(self.\_\_a, self.\_\_b, self.\_\_c)

def check\_numbers(self, a: int, b: int, c: int) -> bool: # Проверяет, изменились ли значения

if (self.\_\_a, self.\_\_b, self.\_\_c) != (a, b, c):

self.\_\_a, self.\_\_b, self.\_\_c = a, b, c

self.process\_numbers()

return True

return False

def process\_numbers(self): # Подгоняет все числа под правила

self.\_\_a = self.\_\_low\_limit if self.\_\_a < self.\_\_low\_limit else self.\_\_a

self.\_\_a = self.\_\_up\_limit if self.\_\_a > self.\_\_up\_limit else self.\_\_a

self.\_\_c = self.\_\_up\_limit if self.\_\_c > self.\_\_up\_limit else self.\_\_c

self.\_\_c = self.\_\_low\_limit if self.\_\_c < self.\_\_low\_limit else self.\_\_c

if not(self.\_\_a <= self.\_\_b <= self.\_\_c):

if self.\_\_a > self.\_\_b:

if self.\_\_a > self.\_\_c:

self.\_\_c = self.\_\_a

self.\_\_b = self.\_\_a

if self.\_\_c < self.\_\_b:

if self.\_\_a > self.\_\_c:

self.\_\_a = self.\_\_c

self.\_\_b = self.\_\_c

self.reset\_numbers()

def \_\_del\_\_(self):

with open(self.settings\_path, "w") as f:

json.dump((self.\_\_a, self.\_\_b, self.\_\_c), f)

class Controller(ctk.CTkFrame):

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

super().\_\_init\_\_(master)

self.rowconfigure((0,1,2,3), weight=1)

self.columnconfigure((0,1,2), weight=1)

self.model = Model(self.set\_numbers)

self.\_\_valid = self.register(self.\_\_validate)

self.\_\_create\_widgets()

self.model.reset\_numbers()

def \_\_create\_widgets(self): # Создание виджетов и подключение событий к функциям

self.label = ctk.CTkLabel(master=self, text="A => B => C", fg\_color="transparent", font=("Times New Roman", 95))

self.label.grid(row=0, column=0, sticky="nsew", columnspan=3)

self.A\_entry = ctk.CTkEntry(master=self, height=40, validate="all", validatecommand=(self.\_\_valid, "%P"))

self.A\_entry.grid(row=1, column=0, padx=15, pady=15, sticky="new")

self.A\_entry.bind("<FocusOut>", self.get\_numbers)

self.A\_entry.bind("<Return>", self.get\_numbers)

self.B\_entry = ctk.CTkEntry(master=self, height=40, validate="all", validatecommand=(self.\_\_valid, "%P"))

self.B\_entry.grid(row=1, column=1, padx=15, pady=15, sticky="new")

self.B\_entry.bind("<FocusOut>", self.get\_numbers)

self.B\_entry.bind("<Return>", self.get\_numbers)

self.C\_entry = ctk.CTkEntry(master=self, height=40, validate="all", validatecommand=(self.\_\_valid, "%P"))

self.C\_entry.grid(row=1, column=2, padx=15, pady=15, sticky="new")

self.C\_entry.bind("<FocusOut>", self.get\_numbers)

self.C\_entry.bind("<Return>", self.get\_numbers)

self.A\_spinbox = sb.Spinbox(master=self, width=150, step\_size=1, validate\_registration=self.\_\_valid)

self.A\_spinbox.grid(row=2, column=0, padx=15, pady=15, sticky="new")

self.A\_spinbox.entry.bind("<FocusOut>", self.get\_numbers)

self.A\_spinbox.entry.bind("<Return>", self.get\_numbers)

self.A\_spinbox.add\_button.bind("<Button-1>", self.get\_numbers)

self.A\_spinbox.subtract\_button.bind("<Button-1>", self.get\_numbers)

self.B\_spinbox = sb.Spinbox(master=self, width=150, step\_size=1, validate\_registration=self.\_\_valid)

self.B\_spinbox.grid(row=2, column=1, padx=15, pady=15, sticky="new")

self.B\_spinbox.entry.bind("<FocusOut>", self.get\_numbers)

self.B\_spinbox.entry.bind("<Return>", self.get\_numbers)

self.B\_spinbox.add\_button.bind("<Button-1>", self.get\_numbers)

self.B\_spinbox.subtract\_button.bind("<Button-1>", self.get\_numbers)

self.C\_spinbox = sb.Spinbox(master=self, width=150, step\_size=1, validate\_registration=self.\_\_valid)

self.C\_spinbox.grid(row=2, column=2, padx=15, pady=15, sticky="new")

self.C\_spinbox.entry.bind("<FocusOut>", self.get\_numbers)

self.C\_spinbox.entry.bind("<Return>", self.get\_numbers)

self.C\_spinbox.add\_button.bind("<Button-1>", self.get\_numbers)

self.C\_spinbox.subtract\_button.bind("<Button-1>", self.get\_numbers)

self.A\_slider = ctk.CTkSlider(master=self, from\_=0, to=100, number\_of\_steps=100)

self.A\_slider.grid(row=3, column=0, padx=15, pady=15, sticky="new")

self.A\_slider.bind("<Button-1>", self.get\_numbers)

self.A\_slider.bind("<B1-Motion>", self.get\_numbers)

self.B\_slider = ctk.CTkSlider(master=self, from\_=0, to=100, number\_of\_steps=100)

self.B\_slider.grid(row=3, column=1, padx=15, pady=15, sticky="new")

self.B\_slider.bind("<Button-1>", self.get\_numbers)

self.B\_slider.bind("<B1-Motion>", self.get\_numbers)

self.C\_slider = ctk.CTkSlider(master=self, from\_=0, to=100, number\_of\_steps=100)

self.C\_slider.grid(row=3, column=2, padx=15, pady=15, sticky="new")

self.C\_slider.bind("<Button-1>", self.get\_numbers)

self.C\_slider.bind("<B1-Motion>", self.get\_numbers)

def set\_numbers(self, a: int, b: int, c: int): # Вставляет значения a, b, c в виджеты

self.A\_entry.delete(0, END)

self.A\_entry.insert(0, str(a))

self.B\_entry.delete(0, END)

self.B\_entry.insert(0, str(b))

self.C\_entry.delete(0, END)

self.C\_entry.insert(0, str(c))

self.A\_spinbox.set(a)

self.B\_spinbox.set(b)

self.C\_spinbox.set(c)

self.A\_slider.set(a)

self.B\_slider.set(b)

self.C\_slider.set(c)

def get\_numbers(self, \*args) -> None: # Получает значения a, b, c из виджетов и отправляет их на обработку в Model

# Если какая либо строка удалена, восстанавливает значения

if "" in [self.A\_entry.get(), self.B\_entry.get(), self.C\_entry.get(), self.A\_spinbox.entry.get(),

self.B\_spinbox.entry.get(), self.C\_spinbox.entry.get()]:

self.model.reset\_numbers()

return

# Ищет нововведённое значение из всех полей ввода (если такого нет, то вставляет первое попавшееся)

a = [int(self.A\_entry.get()), int(self.A\_spinbox.get()), int(self.A\_slider.get())]

unique\_a = list(set(filter(lambda x: a.count(x) == 1, a)))

final\_a = unique\_a[0] if len(unique\_a) > 0 else a[0]

b = [int(self.B\_entry.get()), int(self.B\_spinbox.get()), int(self.B\_slider.get())]

unique\_b = list(set(filter(lambda x: b.count(x) == 1, b)))

final\_b = unique\_b[0] if len(unique\_b) > 0 else b[0]

c = [int(self.C\_entry.get()), int(self.C\_spinbox.get()), int(self.C\_slider.get())]

unique\_c = list(set(filter(lambda x: c.count(x) == 1, c)))

final\_c = unique\_c[0] if len(unique\_c) > 0 else c[0]

self.model.check\_numbers(final\_a, final\_b, final\_c)

def \_\_validate(self, P: str): # Проверяет, что введённое значение является числом

return str.isdigit(P) or P == "" or (P == ("-"+P[1:]) and str.isdigit(P[1:]))

class App(ctk.CTk):

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

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

self.geometry("600x350")

self.title("Лабораторная работа №3.2")

self.grid\_rowconfigure(0, weight=1)

self.grid\_columnconfigure(0, weight=1)

self.controller = Controller(master=self)

self.controller.grid(row=0, column=0, sticky="nsew")

self.protocol("WM\_DELETE\_WINDOW", self.on\_close)

def on\_close(self):

self.controller.model.\_\_del\_\_()

self.destroy()

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

app = App()

app.mainloop()

Файл spinbox.py: <https://github.com/qvap/OOP--Lab-3.2/blob/main/spinbox.py>

Исходный код:

import customtkinter

from tkinter import END

from typing import Callable, Union, Optional

class Spinbox(customtkinter.CTkFrame):

def \_\_init\_\_(self, \*args,

width: int = 100,

height: int = 32,

step\_size: Union[int, float] = 1,

command: Callable = None, validate\_registration,

\*\*kwargs):

super().\_\_init\_\_(\*args, width=width, height=height, \*\*kwargs)

self.step\_size = step\_size

self.command = command

self.configure(fg\_color=("gray78", "gray28")) # set frame color

self.grid\_columnconfigure((0, 2), weight=0) # buttons don't expand

self.grid\_columnconfigure(1, weight=1) # entry expands

self.subtract\_button = customtkinter.CTkButton(self, text="-", width=height-6, height=height-6,

command=self.subtract\_button\_callback)

self.subtract\_button.grid(row=0, column=0, padx=(3, 0), pady=3)

self.entry = customtkinter.CTkEntry(self, width=width-(2\*height), height=height-6, border\_width=0, validate="all", validatecommand=(validate\_registration, "%P"))

self.entry.grid(row=0, column=1, columnspan=1, padx=3, pady=3, sticky="ew")

self.add\_button = customtkinter.CTkButton(self, text="+", width=height-6, height=height-6,

command=self.add\_button\_callback)

self.add\_button.grid(row=0, column=2, padx=(0, 3), pady=3)

# default value

self.entry.insert(0, "0")

def add\_button\_callback(self):

if self.command is not None:

self.command()

try:

value = int(self.entry.get()) + self.step\_size

self.entry.delete(0, END)

self.entry.insert(0, value)

except ValueError:

return

def subtract\_button\_callback(self):

if self.command is not None:

self.command()

try:

value = int(self.entry.get()) - self.step\_size

self.entry.delete(0, END)

self.entry.insert(0, value)

except ValueError:

return

def get(self) -> str:

return self.entry.get()

def set(self, value: int):

self.entry.delete(0, END)

self.entry.insert(0, str(value))