Python 程序设计

第一次作业：面向对象专题

2021211306 班 | 杜抒泽 2021211110

2023 年 10 月 23 日

目录

1. 作业题目 1
2. 作业内容 1
3. 代码说明 7
3.1. 程序基本介绍与运行方式 7
3.2. 程序设计文档：以面向对象为核心 7
3.3. 程序风格：遵循 PEP8 编码规范，实现完备类型注解 9
3.4. 用户使用文档 10
3.5. 总结 12

1. 作业题目

每人独立设计并实现一个小型 Python 程序（功能不限），代码需要涉及：`class`类、对象实例化、继承（分别定义父类和子类）、对象方法（`self`参数）、类方法（`@classmethod`）、静态方法（`@staticmethod`）、对象属性、类属性、多态。

2. 作业内容

程序源代码嵌入下方的 code block 中。**由于代码较长，建议直接点击目录中第三节的超链接跳转代码说明开始阅读。**

```
#! /usr/bin/env python3
# -*- coding: utf-8 -*-
```

```python
import tkinter as tk
from tkinter import simpledialog
from typing import List, Optional


class Color:  # 类 1
    class ColorError(Exception):  # 类 2, 继承 1
        def __init__(self, color: str):
            super().__init__(f'"{color}" is not a valid color.')

    DEFAULT = '#ffffff'  # 静态属性

    def __init__(self, r: int, g: int, b: int):
     if not (self.valid_u8(r) and self.valid_u8(g) and self.valid_u8(b)):
            raise self.ColorError(f'rgb({r}, {g}, {b})')
        self.r = r
        self.g = g
        self.b = b

    @classmethod  # class method 1
    def from_hex(cls, hex_str: str) -> 'Color':
        try:
            if hex_str.startswith('#'):
                hex_str = hex_str[1:]
            r: int = int(hex_str[:2], 16)
            g: int = int(hex_str[2:4], 16)
            b: int = int(hex_str[4:], 16)
        except ValueError:
            raise cls.ColorError(f'#{hex_str}')

        return cls(r, g, b)

    @staticmethod  # static method 1
    def default() -> 'Color':
        return Color.from_hex(Color.DEFAULT)

    @staticmethod  # static method 2
    def valid_u8(value: int) -> bool:
        return 0 ⩽ value ⩽ 255

    def __str__(self) -> str:
        return f'#{self.r:02x}{self.g:02x}{self.b:02x}'
```

```
    def __repr__(self) -> str:
        return f'Color({self.r}, {self.g}, {self.b})'


class Point:  # 类 3
    def __init__(self, x: float, y: float):
        self.x: float = x
        self.y: float = y

    def __repr__(self) -> str:
        return f'Point({self.x}, {self.y})'


class Shape:  # 类 4
    def __init__(self, canvas: Optional[tk.Canvas]):
        self.canvas: Optional[tk.Canvas] = canvas
        self.color: Optional[Color] = None

    def draw(self) -> None:
        raise NotImplementedError(
            'The `draw` method must be implemented by subclasses.')


class Rectangle(Shape):  # 类 5, 继承 2
    def __init__(self, canvas: Optional[tk.Canvas], p1: Point, p2: Point,
color: Optional[Color] = None):
        super().__init__(canvas)
        self.p1: Point = p1
        self.p2: Point = p2
        self.color: Optional[Color] = color

    def draw(self) -> None:
        if self.canvas is None:
            raise ValueError('The `canvas` property must be set.')

        self.canvas.create_rectangle(
         self.p1.x, self.p1.y, self.p2.x, self.p2.y, fill=str(self.color
if self.color else Color.default()))


class Circle(Shape):  # 类 6, 继承 3
```

```
    def __init__(self, canvas: Optional[tk.Canvas], center: Point, radius:
float, color: Optional[Color] = None):
        super().__init__(canvas)
        self.center: Point = center
        self.radius: float = radius
        self.color: Optional[Color] = color

    @classmethod
    def with_two_points(cls, canvas: Optional[tk.Canvas], center: Point,
side: Point, color: Optional[Color] = None) -> 'Circle':
            radius: float = ((center.x - side.x)**2 + (center.y -
side.y)**2)**0.5
        return cls(canvas, center, radius, color)

    def draw(self) -> None:
        if self.canvas is None:
            raise ValueError('The `canvas` property must be set.')

        x0: float = self.center.x - self.radius
        y0: float = self.center.y - self.radius
        x1: float = self.center.x + self.radius
        y1: float = self.center.y + self.radius
        self.canvas.create_oval(
               x0, y0, x1, y1, fill=str(self.color if self.color else
Color.default()))


class NewShapeDialog(simpledialog.Dialog):
    def body(self, master: tk.Tk) -> tk.Entry:  # type: ignore[override]
        self.result: Optional[Shape] = None
         tk.Label(master, text="Shape (circle/rectangle):").grid(row=0,
sticky=tk.W)
      tk.Label(master, text="Color (#rrggbb):").grid(row=1, sticky=tk.W)
       tk.Label(master, text="Point 1 (x, y):").grid(row=2, sticky=tk.W)
       tk.Label(master, text="Point 2 (x, y):").grid(row=3, sticky=tk.W)

        self.shape_entry: tk.Entry = tk.Entry(master)
        self.color_entry: tk.Entry = tk.Entry(master)
        self.point1_entry: tk.Entry = tk.Entry(master)
        self.point2_entry: tk.Entry = tk.Entry(master)

        self.shape_entry.grid(row=0, column=1)
```

```
        self.color_entry.grid(row=1, column=1)
        self.point1_entry.grid(row=2, column=1)
        self.point2_entry.grid(row=3, column=1)

        return self.shape_entry

    def apply(self) -> None:
        shape = self.shape_entry.get()
        color_hex = self.color_entry.get()
        point1_str = self.point1_entry.get()
        point2_str = self.point2_entry.get()

        try:
                            point1 = Point(*map(float, map(str.strip,
point1_str.split(','))))
                            point2 = Point(*map(float, map(str.strip,
point2_str.split(','))))
            color = Color.from_hex(color_hex)
            if 'circle'.find(shape) == 0:
                self.result = Circle.with_two_points(
                    None, point1, point2, color)
            elif 'rectangle'.find(shape) == 0:
                self.result = Rectangle(
                    None, point1, point2, color)
            else:
                raise ValueError(f"Invalid shape: {shape}")
        except Exception as e:
            print(f"An error occurred: {e}")
            self.result = None


class Display:
    def __init__(self, root: tk.Tk):
        self.root: tk.Tk = root
             self.canvas: tk.Canvas = tk.Canvas(self.root, width=600,
height=600)
        self.canvas.pack()
        self.shapes: List[Shape] = []

        self.add_controls()

    def add_shape(self, shape: Shape) -> None:
```

```
        self.shapes.append(shape)
        shape.draw()  # 多态；动态分发

    def clear_all_shapes(self) -> None:
        self.shapes.clear()
        self.canvas.delete("all")

    def prompt_for_shape(self) -> None:
        dialog: NewShapeDialog = NewShapeDialog(self.root)
        shape: Optional[Shape] = dialog.result

        if shape:
            shape.canvas = self.canvas
            self.add_shape(shape)

    def add_controls(self) -> None:
        button_frame: tk.Frame = tk.Frame(self.root)
        button_frame.pack(fill=tk.X)

        clear_button: tk.Button = tk.Button(
          button_frame, text="Clear All", command=self.clear_all_shapes)
        clear_button.pack(side=tk.RIGHT)

        add_shape_button: tk.Button = tk.Button(
          button_frame, text="Add Shape", command=self.prompt_for_shape)
        add_shape_button.pack(side=tk.RIGHT)


def main() -> None:
    root: tk.Tk = tk.Tk()
    root.title('Shapes Display')

    display: Display = Display(root)

    rect: Shape = Rectangle(display.canvas, Point(150, 150),
                            Point(250, 200), Color.from_hex('#8866ff'))
    circ: Shape = Circle(display.canvas, Point(300, 300),
                         50, Color.from_hex('#ff6688'))
    display.add_shape(rect)
    display.add_shape(circ)

    root.mainloop()
```

```
if __name__ == "__main__":
    main()
```

3. 代码说明

3.1. 程序基本介绍与运行方式

这段代码基于 Tkinter 实现了一个简单的 GUI 图形画布，允许用户在画布上绘制不同颜色的矩形和圆形。用户可以通过弹出的对话框输入形状参数，如坐标点和颜色。程序具有添加形状和清除所有形状的功能，为用户提供了直观的图形操作体验。

Tkinter 是 Tcl/Tk GUI 工具包的接口，是 Python 标准库的一部分，会随着 CPython 的安装一同安装。因此，本程序不需要额外安装任何依赖，可以直接运行。若保存上述代码为 `./1st_oop.py`，则可直接在当前工作目录下运行 `$ python3 ./1st_oop.py`。GUI 界面会自行打开。

程序实现了较完善的用户输入错误处理，但由于实现 GUI 报错需要太大代码量且与本次程序设计关系不大，多数错误只会在标准输出流和标准错误流中打印，而在 GUI 中表现为程序无任何变化。

3.2. 程序设计文档：以面向对象为核心

本次作业的主题是面向对象，要求代码涉及

1. 类
2. 对象实例化
3. 继承（自行定义的父类和子类）
4. 对象方法（`self` 参数）
5. 类方法（`@classmethod`）
6. 静态方法（`@staticmethod`）
7. 对象属性
8. 类属性
9. 多态

中的所有方面。以下将就程序中涉及的所有上述 OOP 概念分别进行说明。

3.2.1. 类的设计概述

本程序中共设计了 6 个类，其中 5 个类是用户可见的，1 个类是用户不可见的。这 6 个类的关系如下表：

类	父类（除 Object）	包含的组合类	说明
Color			24 位 RGB 颜色类
ColorError	Exception		颜色类的异常类
Point			平面坐标点类
Shape		Color	平面形状类（抽象类）
Rectangle	Shape	Point, Color	矩形类
Circle	Shape	Point, Color	圆形类
NewShapeDialog	simpledialog.Dialog		对话框类，用于添加图形时弹出对话框以获取用户输入

由此可见要求中的 [1. 类] 和 [3. 继承] 在程序中得到了体现。

3.2.2. Color 和 ColorError

Color 类是本程序中的基础类，它用于表示 24 位 RGB 颜色。Color 类实现了构造器 `__init__` 和魔术方法 `__str__`、`__repr__` 用于可能的字符串输出。此外，它实现了如下的属性和方法：

- 类属性 `DEFAULT = '#ffffff'`

  表示默认颜色（白色）。

- 对象属性 `r: int, g: int, b: int`

  表示 24 位 RGB 颜色的三个分量，均为 8 位无符号整数。

- 类方法 `@classmethod from_hex(cls, hex_str: str) -> 'Color'`

  接受一个 `str` 类型的参数，表示 24 位 RGB 颜色的十六进制表示。若参数不是合法的十六进制表示，则抛出 ColorError 异常。

- 类方法 `@classmethod default() -> 'Color'`

  构造默认颜色（白色）对象实例并返回。

- 静态方法 `@staticmethod valid_u8(value: int) -> bool`

接受一个 int 类型的参数，表示 8 位无符号整数。若参数不是合法的 8 位无符号整数，则返回 False，否则返回 True。

此外，Color 类还定义了一个内部类 ColorError，用于表示颜色类的异常。ColorError 类继承自 Exception 类，因此它是一个异常类。ColorError 类实现了构造器 __init__，用于接受一个 str 类型的参数，表示异常信息。ColorError 类还实现了魔术方法 __str__、__repr__ 用于可能的字符串输出。

由此可见要求中的 [2. 对象实例化]、[4. 对象方法]、[5. 类方法]、[6. 静态方法]、[7. 类属性]、[8. 对象属性] 在程序中得到了体现。

3.2.3. Shape、Rectangle 和 Circle

这三个类是本程序的主体。

Shape 类是一个抽象类，它定义了一个抽象方法 draw，用于绘制形状。Shape 类还定义了对象属性 canvas: tk.Canvas，表示形状所在的画布和 color: Optional[Color] 表示形状的颜色。

Rectangle 类和 Circle 类都继承自 Shape 类，因此它们都需要实现 draw 方法。Rectangle 类和 Circle 类都实现了魔术方法 __repr__，用于可能的字符串输出。

Circle 特别实现了一个类方法 @classmethod def with_two_points(...)，用于接受两个 Point 类型的参数，分别表示圆心和圆周上的一点，然后计算圆的半径并返回一个 Circle 类型的对象实例。

此处的 draw 方法在后续实现了动态分发。在 class Display 的方法 add_shape 中，shape 被类型注解为 Shape，在之上施用了 shape.draw()，这里的 draw 方法不会调用抽象基类的方法，而是会根据 shape 的实际类型（Rectangle 或 Circle）进行动态分发，实现多态。

由此可见要求中的 [9. 多态] 在程序中得到了体现。

程序完成了作业题目中的所有要求。

3.3. 程序风格：遵循 PEP8 编码规范，实现完备类型注解

这段代码虽然严格遵守 Python 的 PEP8 编码规范，但观感实际上并不是很好，看起来十分啰嗦。这很大程度上是因为代码实现了冗长的类型注解。

利用 mypy 分析工具，开启所有不允许偷懒的选项，我们得到如下的类型检查结果：

```
$ mypy 1st_oop.py --check-untyped-defs --disallow-untyped-calls --
disallow-untyped-defs --disallow-incomplete-defs --disallow-untyped-
decorators
Success: no issues found in 1 source file
```

可以看到，这段代码在类型检查上是完全正确的。美观性的略微丧失并未导致可读性的降低，但是带来了类型安全，这种 trade-off 是完全值得的。

3.4. 用户使用文档

打开程序后，界面如下图：

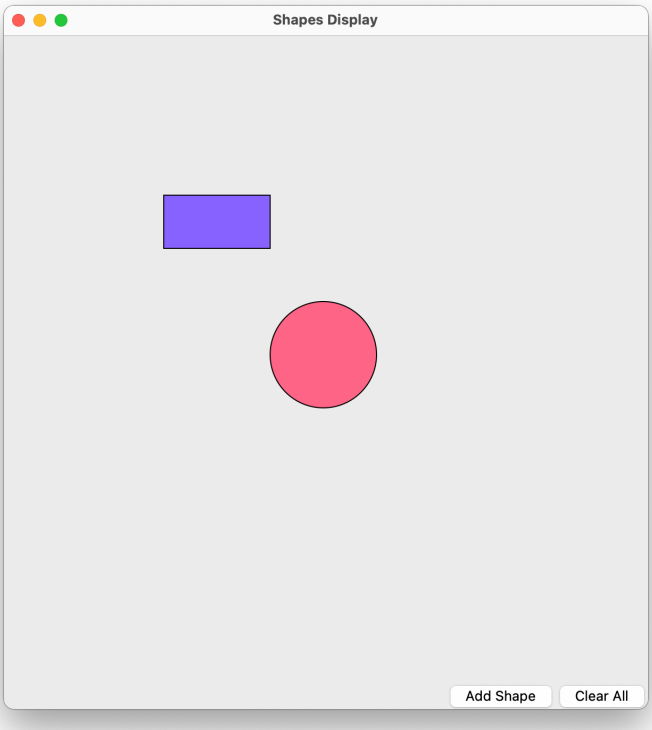

图 1: 程序主界面

这是一块 600 × 600 像素的画布，画布上有两个预置的图形。用户可以点击 Add Shape 按钮添加新的图形，点击 Clear All 按钮清除所有图形。

点击 Add Shape 弹出的对话框如下图：

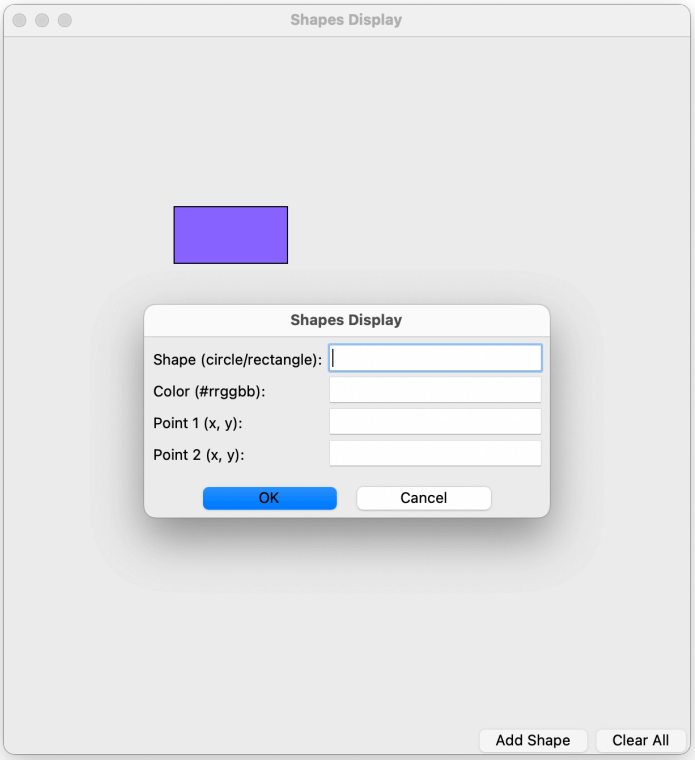

图 2: 添加图形对话框

对话框有较强的引导性。其中第一个输入框只要输入 circle 或 rectangle 的任意子串即可识别。第二个输入框输入格式为 `^#?[0-9a-fA-F]{6}$`。第三个和第四个输入框输入格式为用逗号隔开的两个浮点数（逗号后可以有空格）。若用户输入不合法，则会在标准输出流和标准错误流中打印错误信息，但不会在 GUI 中表现出来。

对于矩形，输入的两个坐标为矩形的对角线的两个端点。对于圆形，输入的两个坐标为圆心和圆周上的一点。

例如，我们不清除原有的图形，直接输入以下三组数据：

```
circ
#f3f0bc
200, 400
200, 450
```

```
rect
E9f1Fa
400, 200
500, 250
```

```
r
#4d9BAb
450, 450
550, 550
```

会得到如下的画板：

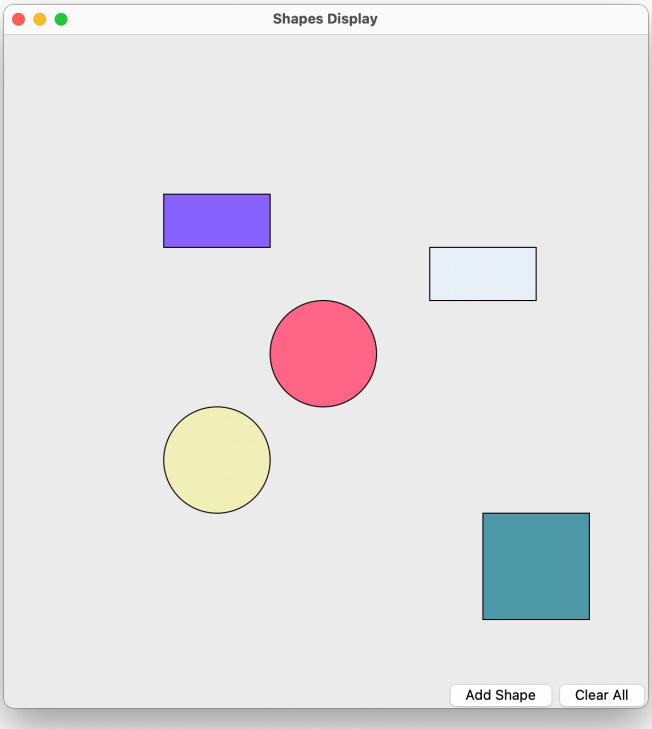

图 3: 添加图形后的画板

这充分说明该画板实现了应有的功能，且具有较好的用户交互界面。

3.5. 总结

本次作业成功地展示了面向对象编程范式的实用性和效率。通过构建一个图形绘制应用，我实际应用了类、组合、继承、多态等核心概念，进行严格的类型检查。通过这词作业，我体会到了面向对象编程在处理复杂问题时的直观性和灵活性，确认了其在现代编程实践中不可或缺的地位。