# Подробная структура кода погодного приложения

## 1. Импорт библиотек

Импортируются модули, необходимые для GUI, работы с файлами и абстракциями.

import tkinter as tk  
from tkinter import ttk, messagebox  
from abc import ABC, abstractmethod  
import json  
import os

## 2. Цветовые градиенты

Определение цветов для различных погодных условий.

weather\_gradients = {  
 "clear": ("#FFD700", "#FFFFFF"),  
 "rain": ("#4682B4", "#87CEFA"),  
 ...  
}

## 3. Советы по погоде

Хранит советы в зависимости от погодных условий.

weather\_tips = {  
 "clear": "Ясная погода...",  
 "snow": "Снег..."  
}

## 4. Иконки погоды

Символические иконки для визуального отображения погоды.

weather\_icons = {  
 "clear": "☀️",  
 "rain": "🌧️",  
 ...  
}

## 5. Названия месяцев

Словарь для перевода числового формата месяца в текстовый.

month\_names = {  
 "04": "апреля",  
 "05": "мая",  
 ...  
}

## 6. Абстрактный класс BaseWeatherApp

Базовый интерфейс для главного окна приложения.

class BaseWeatherApp(ABC):  
 def \_\_init\_\_(self, root):  
 self.root = root  
 @abstractmethod  
 def update\_weather(self): ...

## 7. Класс WeatherDataHandler: загрузка данных

Загружает и хранит данные о погоде из JSON-файла.

class WeatherDataHandler:  
 def load\_weather\_data(self):  
 with open(...) as f:  
 return json.load(f)

## 8. WeatherDataHandler: получение текущей погоды

Метод возвращает текущую погоду по городу.

def get\_current\_weather(self, city):  
 return self.weather\_data.get(city, {}).get("current", {...})

## 9. WeatherDataHandler: форматирование даты

Преобразует дату в формат вида '15 апреля'.

def format\_date(self, date):  
 day, month = date.split('-')[2], date.split('-')[1]  
 return f"{int(day)} {month\_names[month]}"

## 10. Абстрактный UI-компонент

Базовый класс для графических компонентов (рендер, очистка).

class WeatherUIComponent(ABC):  
 def clear\_frame(self, frame):  
 for widget in frame.winfo\_children(): widget.destroy()

## 11. Компонент CityListComponent

Отображает список городов и обрабатывает выбор.

class CityListComponent(WeatherUIComponent):  
 def render(self):  
 for city in self.cities:  
 label = tk.Label(...)  
 label.bind("<Button-1>", ...)

## 12. События наведения на город

Изменение стиля при наведении мыши на город.

def on\_enter(self, label): label.config(bg="#4A4A4A")  
def on\_leave(self, label): label.config(bg="#2C2C2C")

## 13. Компонент WeatherDisplayComponent

Показывает текущую погоду, почасовой и 14-дневный прогноз.

class WeatherDisplayComponent(WeatherUIComponent):  
 def render(self, city):  
 self.show\_current(city)  
 self.show\_hourly\_forecast(city)

## 14. Отображение текущей погоды

Создает блок с текущими условиями и подробностями.

def show\_current(self, city):  
 data = self.data\_handler.get\_current\_weather(city)  
 tk.Label(..., text=f"+{data['temp']}°")

## 15. Почасовой прогноз

Отображает температуру и осадки по времени суток.

def show\_hourly\_forecast(self, city):  
 for time, data in hourly\_data.items():  
 tk.Label(..., text=time)

## 16. Прогноз на 14 дней

Показывает иконки и температуру на каждый из дней.

def show\_14\_day\_forecast(self, city):  
 for i, date in enumerate(dates):  
 tk.Label(..., text=formatted\_date)

## 17. Главный класс приложения WeatherApp

Главное окно, инициализация и связь компонентов.

class WeatherApp(BaseWeatherApp):  
 def \_\_init\_\_(self, root):  
 self.data\_handler = WeatherDataHandler()

## 18. UI-настройка и запуск компонентов

Создание фреймов и инициализация элементов интерфейса.

def setup\_ui(self):  
 self.city\_list = CityListComponent(...)  
 self.weather\_display = WeatherDisplayComponent(...)

## 19. Обновление погоды

Обновляет интерфейс при выборе города.

def update\_weather(self):  
 self.weather\_display.render(self.current\_city)

## 20. Запуск tkinter

Создание окна и старт главного цикла программы.

if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 app = WeatherApp(root)  
 root.mainloop()