# Методы ООП в проекте погодного приложения

**Методы ООП в проекте погодного приложения на Python**

В данном проекте реализованы основные методы объектно-ориентированного программирования (ООП): \*\*инкапсуляция\*\*, \*\*наследование\*\*, \*\*полиморфизм\*\*, \*\*абстракция\*\*. Рассмотрим каждый метод с пояснением, примером и фрагментом кода из `pogoda.py`.

---

## 1. Инкапсуляция

\*\*Описание:\*\* Сокрытие деталей реализации класса от внешнего мира. Все данные и методы находятся внутри классов и управляются через интерфейс (методы).

**Пример в проекте:**

Класс `WeatherDataHandler` инкапсулирует всю работу с JSON-файлом и предоставляет только нужные методы: `get\_current\_weather`, `get\_hourly\_forecast`, `get\_daily\_forecast`.

**Код:**

class WeatherDataHandler:  
 def \_\_init\_\_(self, data\_file="weather\_data.json"):  
 self.data\_file = data\_file  
 self.weather\_data = self.load\_weather\_data()  
  
 def load\_weather\_data(self):  
 # Загрузка данных из JSON файла  
 ...  
  
 def get\_current\_weather(self, city):  
 return self.weather\_data.get(city, {}).get("current", {...})

**Использование:**

self.data\_handler = WeatherDataHandler()  
data = self.data\_handler.get\_current\_weather("Москва")

## 2. Наследование

**Пример в проекте:**

**Код:**

class WeatherApp(BaseWeatherApp):  
 def \_\_init\_\_(self, root):  
 super().\_\_init\_\_(root)  
 ...

**Использование:**

app = WeatherApp(root)

## 3. Полиморфизм

**Пример в проекте:**

**Код:**

class WeatherUIComponent(ABC):  
 @abstractmethod  
 def render(self):  
 pass  
  
class CityListComponent(WeatherUIComponent):  
 def render(self):  
 # Отрисовка списка городов  
 ...  
  
class WeatherDisplayComponent(WeatherUIComponent):  
 def render(self, city):  
 # Отрисовка погодных данных  
 ...

**Использование:**

self.city\_list.render()  
self.weather\_display.render("Москва")

## 4. Абстракция

**Пример в проекте:**

**Код:**

class BaseWeatherApp(ABC):  
 @abstractmethod  
 def update\_weather(self):  
 pass  
  
class WeatherUIComponent(ABC):  
 @abstractmethod  
 def render(self):  
 pass

**Использование:**

class WeatherApp(BaseWeatherApp):  
 def update\_weather(self):  
 # Обновление погодной информации  
 ...