Dip_Act35

🔥 DIP Activity 35: Refactor the Weather Alert System

🧠 Scenario Recap:

The current `WeatherAlert` class is tightly coupled with `EmailSender`. If we want to use SMS or another alert method in the future, we'd have to edit the `WeatherAlert` class — which violates **DIP**.

Code block

```
# Low-level class
class EmailSender:
    def send_email(self, message):
        print(f"Sending email: {message}")

# High-level module
class WeatherAlert:
    def __init__(self):
        self.email_sender = EmailSender()  # ❌ tightly coupled to EmailSender

    def send_alert(self, condition):
        if condition == "storm":
            self.email_sender.send_email("⚠️ Storm warning!")

```

📝 What to Do:

1. Analyze the current dependency between `WeatherAlert` and `EmailSender`.
   What is the **problem with this relationship**?
2. Introduce an **abstraction** for sending messages (you decide what to name it).
   - What method should it have?
   - What do you want different notifiers to do?
3. Create two concrete classes:

    - One that sends emails
    - One that sends SMS
4. They should each follow the behavior defined by your abstraction.
5. Refactor the alert system so that:
    - It doesn’t care *how* the message is sent
    - It works with *any* object that follows your interface

⚡ Bonus Twist:

- Add a new type of notification (like `PushNotifier`)
- Prove that you don’t have to touch the `WeatherAlert` class to support it