TP 1

Integrantes:
Nahuel Matias Benitez
Lucas Efrain Oliaro Vera

En este trabajo modificaremos la implementación actual del Patrón Observer, que utiliza un modelo "Push" (donde el Sujeto envía los datos a los Observadores), para convertirla en un modelo "Pull" (donde el Sujeto solo notifica el cambio, y los Observadores solicitan activamente los datos que necesitan).

Diferencia entre modelos Push y Pull

En el patrón *Observer*, el *Subject* notifica a sus *Observers* cuando algo cambia.

En el modelo ***Push***, el *Subject* empuja los datos nuevos directamente a cada observador a través de los parámetros del método update.

- Ventaja: los observadores reciben toda la información necesaria en la notificación.
- Desventaja: el Subject tiene que decidir qué datos entregar, y puede sobrecargar a los Observers con información que no necesitan.

En el modelo ***Pull***, el *Subject* solo avisa que algo cambió, y cada *Observer* decide **tirar** la información que le interesa usando los getters del *Subject*.

- Ventaja: los *Observers* tienen más control y flexibilidad sobre qué datos consumir.
- Desventaja: cada *Observer* necesita mantener una referencia al *Subject*.

Modificar las interfaces y clases existentes para implementar el modelo Pull

El cambio principal que hicimos fue:

Interfaz Observer: cambiamos la firma de update para que no reciba parámetros.

```
class Observer(ABC):
    @abstractmethod
    def update(self):
        pass
```

Clase WeatherData (Subject concreto):
En `notify_observers`, ya no pasamos `(temperature, humidity, pressure)`, sino que simplemente llamamos a *observer.update()*.

Clases Display (Observers concretos):
Ahora cada display guarda una referencia al `WeatherData` y, dentro de `update()`, llama a los getters *(`get_temperature()`, `get_humidity()`*, etc.) para obtener los datos que realmente le interesan.

Observar cómo cambian las responsabilidades entre el Sujeto y los Observadores

Con el cambio a Pull:

- El **Sujeto (`WeatherData`)** tiene menos responsabilidad: solo notifica cambios. Ya no se preocupa por decidir qué datos deben recibir los observadores.
- Los **Observadores** tienen más responsabilidad: deciden qué información necesitan y la obtienen directamente del *`Subject`*.

Esto produce un **acoplamiento más débil** porque el *`Subject`* no depende de los detalles de cada *`Observer`*.

Relación con el código que implementamos

Lo que hicimos en el código es un reflejo práctico de ese cambio de responsabilidades:
En *WeatherData*, *notify_observers* pasó de:

```
observer.update(self._temperature, self._humidity, self._pressure)
```

actualizado

```
observer.update()
```

En *CurrentConditionsDisplay* (y los demás displays), *update* pasó de recibir los datos directamente a hacer:

```
def update(self):
    self._temperature = self._weather_data.get_temperature()
    self._humidity = self._weather_data.get_humidity()
    self.display()
```

En el *main*, no cambiamos nada: la lógica de notificación funciona igual, pero los observers ahora “tiran” la información que necesitan en lugar de recibirla “empujada”.

Diagrama UML