Instituto Tecnológico y de Estudios Superiores de Occidente – ITESO

ITESO

Universidad Jesuita de Guadalajara

Materia: Diseño de software

Maestro: Pineda Galindo Jose Gerardo

Tarea N1

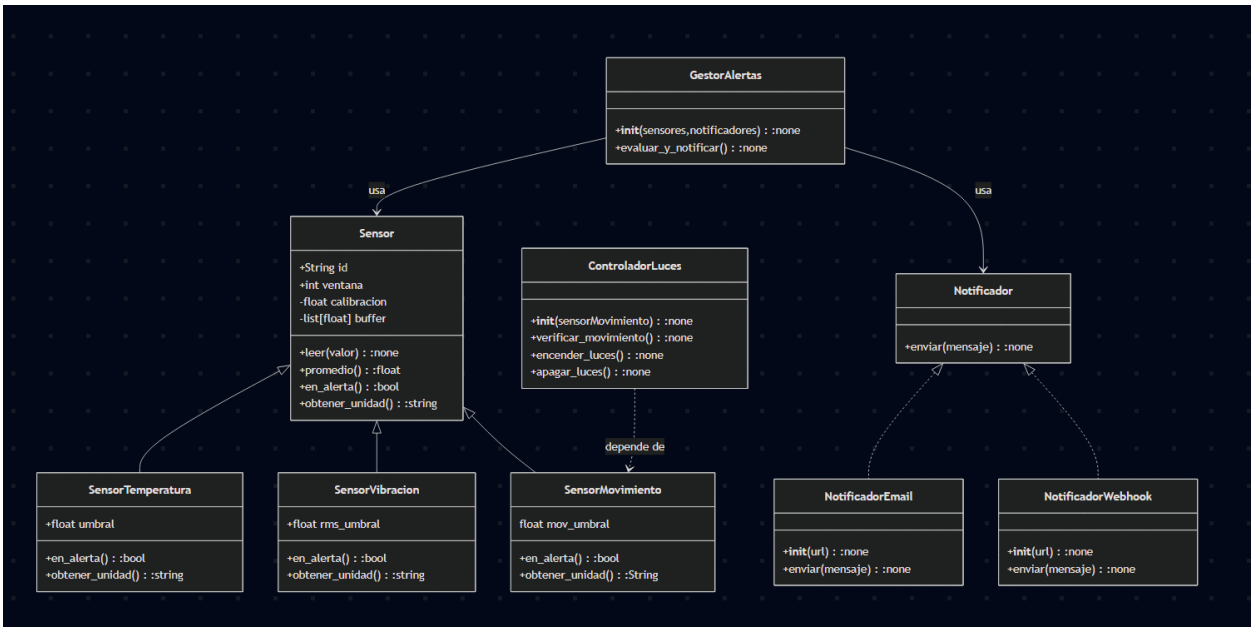
Sesión: 3

Fecha: 08/29/25

Temas: 4 Pilares de la programación orientada o objetos

Autor(es): Lara Valencia Fabricio Daniel

Diagrama UML




```
classDiagram
    Notificador <|.. NotificadorEmail
    Notificador <|.. NotificadorWebhook
    Sensor <|-- SensorTemperatura
    Sensor <|-- SensorVibracion
    Sensor <|-- SensorMovimiento
    GestorAlertas --> Sensor : usa
    GestorAlertas --> Notificador : usa
    ControladorLuces ..> SensorMovimiento : depende de
    class Notificador{
        +enviar(mensaje):none
    }
    class NotificadorEmail{
        +__init__(url):none
        +enviar(mensaje):none
    }
    class NotificadorWebhook{
        +__init__(url):none
        +enviar(mensaje):none
    }
    class Sensor{
        +String id
        +int ventana
        -float calibracion
        -list[float] buffer
        +leer(valor):none
```

```
        +promedio():float
        +en_alerta():bool
        +obtener_unidad():string
    }
    class SensorTemperatura{
        +float umbral
        +en_alerta():bool
        +obtener_unidad():string
    }
    class SensorVibracion{
        +float rms_umbral
        +en_alerta():bool
        +obtener_unidad():string
    }
    class SensorMovimiento{
        float mov_umbral
        +en_alerta():bool
        +obtener_unidad():String
    }
    class GestorAlertas{
        +__init__(sensores,notificadores):none
        +evaluar_y_notificar():none
    }
    class ControladorLuces{
        +__init__(sensorMovimiento):none
        +verificar_movimiento():none
        +encender_luces():none
        +apagar_luces():none
    }
```

Elementos agregados

Herencia:

```python
@dataclass
class SensorMovimento(Sensor):
    mov_umbral: float = 1.0
    def en_alerta(self) -> bool:
        if len(self._buffer) < 2:
            return False
        desviacion_estandar = stdev(self._buffer)
        return desviacion_estandar >= self.mov_umbral
    def obtener_unidad(self) -> str:
```

```
        return "m/s"
```

Metodo Abstracto:

```
@dataclass
class Sensor(ABC):
    id: str
    ventana: int = 5
    _calibracion: float = field(default=0.0, repr=False) # encapsulado
    _buffer: list[float] = field(default_factory=list, repr=False)
    def leer(self, valor: float) -> None:
        """Agrega lectura aplicando calibración y mantiene ventana móvil."""
        v = valor + self._calibracion
        self._buffer.append(v)
        if len(self._buffer) > self.ventana:
            self._buffer.pop(0)

    @property
    def promedio(self) -> float:
        return mean(self._buffer) if self._buffer else 0.0

    @abstractmethod
    def en_alerta(self) -> bool: ...

    @abstractmethod
    def obtener_unidad(self) -> str: ...
```

Subclase:

```
@dataclass
class Sensor(ABC):
```

and

```
@dataclass
class SensorMovimento(Sensor):
```

Dependencia:

```
class GestorAlertas:
    def __init__(self, sensores: List[Sensor], notificadores: List[Notificador])
-> None:
```

```
        self._sensores = sensores
        self._notificadores = notificadores

    def evaluar_y_notificar(self) -> None:
        for s in self._sensores:
            if s.en_alerta():
                msg = f"ALERTA Sensor {s.id} en umbral (avg={s.promedio:.2f})"
                for n in self._notificadores:
                    n.enviar(msg)
```

Agregacion:

```
class GestorAlertas:
    def __init__(self, sensores: List[Sensor], notificadores: List[Notificador])
-> None:
        self._sensores = sensores
        self._notificadores = notificadores

    def evaluar_y_notificar(self) -> None:
        for s in self._sensores:
            if s.en_alerta():
                msg = f"ALERTA Sensor {s.id} en umbral (avg={s.promedio:.2f})"
                for n in self._notificadores:
                    n.enviar(msg)
```