TAREA 3.3 - Creación de un componente propio con QWidget

TAREA 3.3 - Creación de un componente propio con `QWidget`

Componente a desarrollar: `PanelSemaforo`

En esta tarea vas a crear un **componente gráfico personalizado** llamado `PanelSemaforo`, heredado de `QWidget`.

El componente debe simular un **semáforo compacto**, con tres luces: roja, amarilla y verde.

El semáforo cambiará de estado cada vez que se pulse un botón incorporado dentro del propio componente.

1. Objetivo

Aprender a:

- crear un widget propio heredando de `QWidget`,
- gestionar un **estado privado** dentro del componente,
- redibujar usando `paintEvent` y `QPainter`,
- combinar dibujo con widgets internos (`QPushButton`),
- exponer métodos públicos para interactuar con el componente.

2. Funcionamiento requerido

Tu componente `PanelSemaforo` debe cumplir lo siguiente:

✓ Luces del semáforo

- Tres luces dibujadas como **círculos** con `QPainter` (`drawEllipse`).
- Solo **una luz encendida** en cada momento.
- Colores a utilizar:
  - rojo

  - amarillo
  - verde
- Las luces apagadas deben verse en gris.

✓ Cambio de estado

El semáforo debe seguir este ciclo:

1. rojo
2. amarillo
3. verde
4. vuelve a rojo

Cada vez que se pulse el botón "Cambiar", el semáforo debe avanzar al siguiente color.

✓ Estado interno

- Usa un **atributo privado** (`__estado_actual`) para almacenar el color actual.
- Usa valores de texto: `"rojo"`, `"amarillo"`, `"verde"`.

✓ Métodos públicos obligatorios

Tu clase debe incluir:

```
estado()      # devuelve el color actual como cadena
reiniciar()   # vuelve al estado 'rojo'
```

✓ Botón interno

El widget debe incluir un **QPushButton** llamado "Cambiar", situado debajo del dibujo.

✓ Repintado

Cada vez que cambie el estado, debes llamar a `self.update()` para que `paintEvent` vuelva a dibujar el semáforo correctamente.

3. Restricciones técnicas

Para dibujar el semáforo puedes usar **solamente**:

- `drawEllipse`
- `drawRect`
- `setPen`
- `setBrush`
- `setRenderHint(QPainter.Antialiasing)`

4. Estructura recomendada

Tu componente puede organizarse así:

- Un `QVBoxLayout` como diseño principal.
- El área de dibujo irá directamente dentro del propio widget (en `paintEvent`).
- Un botón `QPushButton` abajo del todo.
- Las posiciones de las luces se calcularán dentro de `paintEvent`.

5. Ampliación

Semáforo automático con `QTimer`

Vamos a añadirle una mejora al componente: hacer que los colores del semáforo se vayan **alternando solos** cada cierto tiempo, utilizando un `QTimer`.

✓ Requisitos de la ampliación

- Añade un `QTimer` como atributo del componente.
- Configura el temporizador para que dispare una señal cada *X* milisegundos (por ejemplo, cada 1000 ms = 1 segundo).
- Cada vez que salte el temporizador, el semáforo debe pasar al siguiente estado del ciclo:
  - rojo → amarillo → verde → rojo → …
- El cambio automático debe reutilizar la misma lógica que el botón "Cambiar"

(puedes llamar al mismo método que usas con el botón).

- El temporizador debe empezar a funcionar al crear el componente, o bien al pulsar un botón extra, según lo decidas.

6. Entrega

La entrega debe incluir:

- Archivo `apellido1_nombre_T3.3.py` con comentarios.
- Archivo `apellido1_nombre_T3.3_a.py` con comentarios.

Rúbrica

Criterio	Puntos
Funcionamiento general del componente	3
Cambio correcto entre estados	2
Uso adecuado de `paintEvent` y `QPainter`	4
Claridad del código y explicación	1