Programación Avanzada
IIC2233 2024-1

Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Dante Pinto - Francisca Cattan

Anuncios

1. Hoy tenemos la tercera experiencia.
2. No olviden estar atentos las issues para la T3. Por ejemplo:
    a. Link a la issue
    b. Usar `encoding='latin-1'` para abrir los archivos.
3. Encuesta de Carga Académica ¡No se olviden!

Interfaces gráficas II

Concurrencia en PyQt

- QThread
- QTimer

Concurrencia en PyQt

QThread

- Es un Thread, como el visto anteriormente, pero dentro de PyQt.
- Se utiliza igual que un Thread, es decir, definir un método **run()** y hacer `.start()` del QThread.
- También disponemos de **QMutex**, el análogo al Lock en PyQt. Este nos servirá para asegurar el acceso a un recurso crítico.

Concurrencia en PyQt

QTimer

- Otra herramienta para generar concurrencia (al igual que el QThread).
- Por defecto, se define un tiempo N y cada N milisegundos, se ejecuta la función/método objetivo.
- Podemos hacer **`.stop()`** para detener su siguiente ejecución o usar nuevamente **`.setInterval(N)`** para cambiar el tiempo entre intervalos.

Concurrencia en PyQt

QTimer SingleShot vs NO SingleShot

- **SingleShot:** se ejecuta solo una vez luego de que pase el tiempo indicado en `.setInterval(N).` Análogo a los `Timer` de `Threading.`
- **`No SingleShot`:** se ejecuta periódicamente hasta que se se detenga con `.stop()` o se cierre el programa. Es la configuración por defecto de un QTimer.

- Usamos `.setSingleShot(True)` para que el QTimer sea SingleShot (se **solo 1 vez** luego de los N milisegundos indicados)

Concurrencia en PyQt

QTimer vs QThread

- Ambos son formas distintas de aplicar concurrencia en PyQt.
- Con un poco de creatividad, **ambas clases pueden lograr el mismo efecto**, solo que con código distinto.
- Como son clases, podemos heredar de ellas para crear objeto más personalizado.
- Queda a tu criterio cuál de los 2 usar de aquí en adelante, puedes utilizar únicamente uno o intentar usar ambos. **¡Tú eliges 😁 !**

Concurrencia en PyQt

QTimer vs QThread

- Ambos son formas distintas de aplicar concurrencia en PyQt.
- Con un poco de creatividad, **ambas clases pueden lograr el mismo efecto**, solo que con código distinto.
- Como son clases, podemos heredar de ellas para crear objeto más personalizado.
- Queda a tu criterio cuál de los 2 usar de aquí en adelante, puedes utilizar únicamente uno o intentar usar ambos. **¡Tú eliges 😁 !**
- ***spoiler*:** aunque elijas usar únicamente uno, en el examen podemos preguntar por cualquiera de los 2 o ambos 🙈 .

Misceláneos

- isAutoRepeat
- QMediaPlayer
- QSoundEffect

Misceláneos

isAutoRepeat

- Método de un evento en KeyPressEvent (QKeyEvent)
- Este método retorna un booleano que:
  - Toma el valor de **False** si el evento gatillado fue producto de presionar por **primera** vez una tecla.
  - Toma el valor de **True** si es un evento producto de **mantener** presionada la tecla.
- Útil cuando solo deseamos mandar una señal cuando se presiona la tecla, pero si el usuario la mantiene presionada, no ocurra nada.

Misceláneos

QSoundEffect

- Objeto de PyQt para reproducir archivos **.wav.**

```
self.media_player_wav = QSoundEffect(self) # Creo reproductor
file_url = QUrl.fromLocalFile(join("sounds", "musica.wav")) # Creo un path
media_player_mp3.setSource(file_url) # Indico el path al reproductor
media_player_mp3.play() # Reproducir
```

Misceláneos

QMediaPlayer

- Objeto de PyQt para reproducir archivos **.mp3**.
- Puede salir un *warning* en consola. No te preocupes 😁

```
self.media_player = QMediaPlayer(self) # Creo reproductor
self.media_player.setAudioOutput(QAudioOutput(self)) # Indico que será modo audio
file_url = QUrl.fromLocalFile(join("sounds", "musica.mp3")) # Creo un path
self.media_player.setSource(file_url) # Indico el path al reproductor
self.media_player.play() # Reproducir
```

Diagrama de Clases

- Elemento **visual** para caracterizar **clases** que componen un sistema

- Muestran **atributos**, **comportamientos** y **relaciones** entre clases

Diagrama de Clases

- Existe el formato UML, pero este curso no se regirá 100% por él.
  Se harán algunos cambios para simplificar su uso dentro del curso.

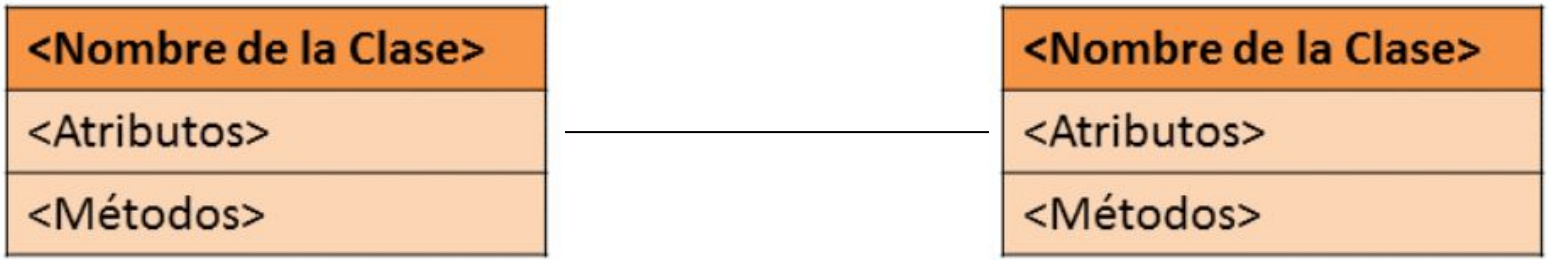


Diagrama de Clases

Formato del curso

- Parte 1: atributos y *property*
  - Nombre
  - Tipo de dato
  - Diferenciar atributo VS *property*

- Parte 2: métodos
  - Nombre
  - Argumentos del método
  - Tipo de dato de su *return*.

Auto	
+	dueño: str or None
+	marca: str
+	modelo: str
+	_kilometraje: int
+	@kilometraje: int (getter y setter)
+	conducir(kms): None
+	vender(dueño): Str

Diagrama de Clases

Relaciones - Herencia

- Se utiliza una flecha para indicar si una clase **hereda** de otra clase.
- La flecha apunta a la clase “padre”
- En este ejemplo, LadrilloMoneda y LadrilloMovil (tipos de ladrillos del juego Super Mario) heredan de Ladrillo.

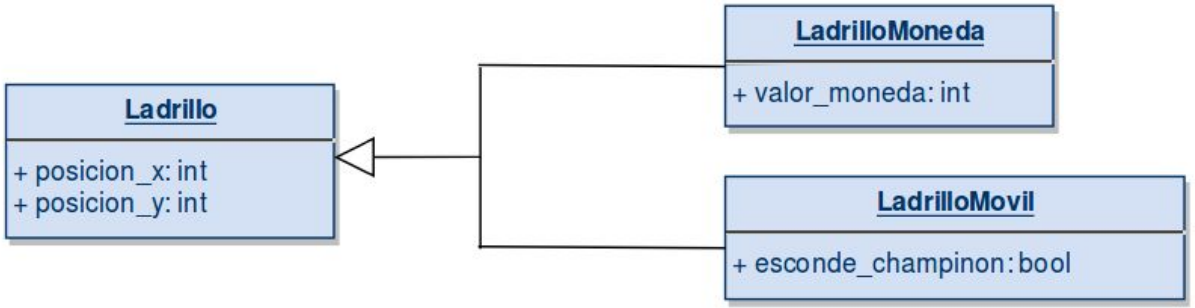
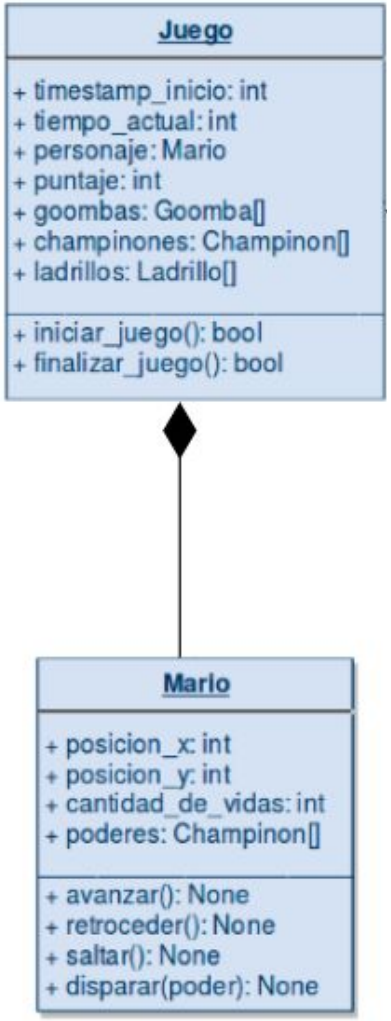


Diagrama de Clases

Relaciones - Contención

- Se utiliza un rombo para indicar si una clase **contiene** a otra clase.
- El rombo está junto a la clase “contenedora”.
- En este ejemplo, la clase Juego **contiene** a la clase Mario.

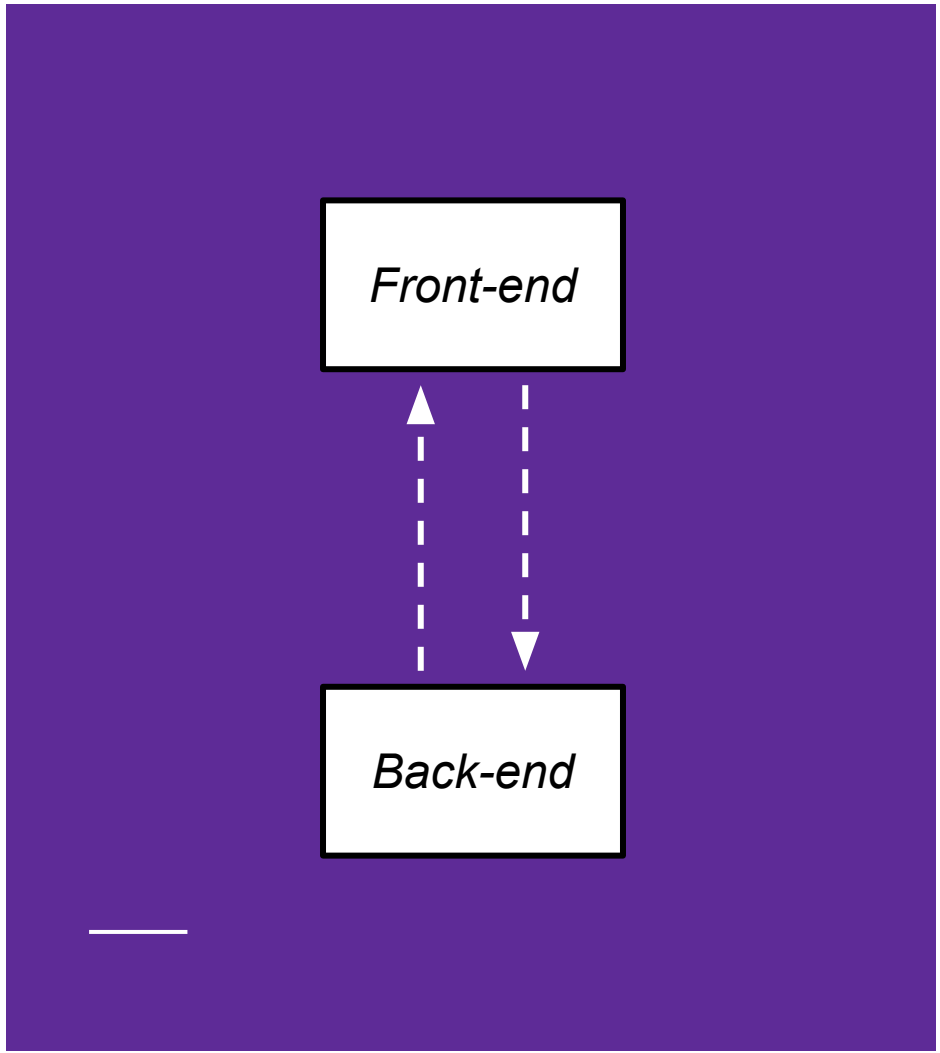


Último mensaje

- Ahora que hay muchas clases en juego.

REPETIMOS

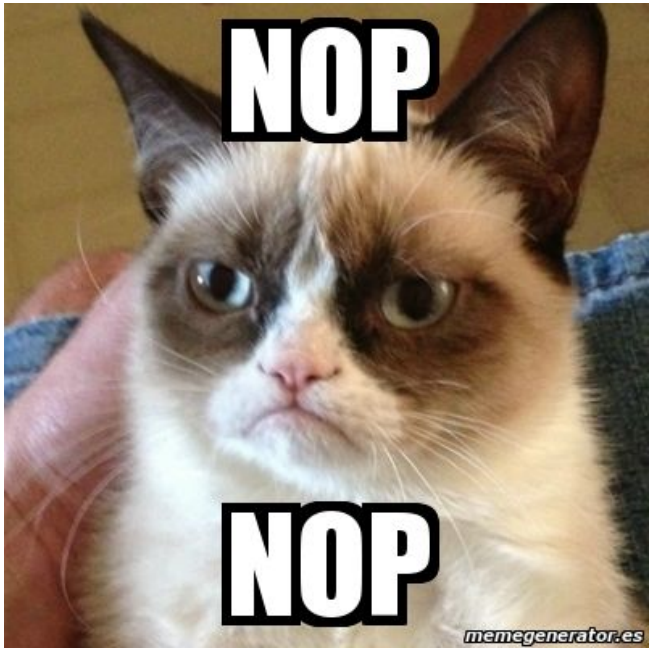
Tengan mucho cuidado con las **dependencias circulares**



Tengan mucho cuidado con las dependencias circulares

```
class Frontend:
     def __init__(self):
          self.backend = Backend(self)


class Backend:
     def __init__(self, frontend):
          self.frontend = frontend
```



Tengan mucho cuidado con las dependencias circulares

```
class Frontend:
    def __init__(self):
        sel
cl
        , frontend):
        self.frontend = frontend
```

¡Usen señales!

Programación Avanzada
IIC2233 2024-1

Hernán Valdivieso - Daniela Concha - - Dante Pinto - Francisca Cattan