Empecemos partiendo de la premisa de que delegar en RegisteredUser la responsabilidad de calcular los totales me parece poco apropiada, ya que difícilmente podemos argumentar que es de su responsabilidad, ateniéndonos al Single Responsability Principle. Dicho esto, mucho mejor si implementaremos un servicio adicional que, provisto del Id de un RegisteredUser, recaba los videos consumidos y sí, devuelve el importe total. Este servicio podría ser consumido por RegisteredUser, así como por otros posibles casos de uso.

De esta manera tenemos un único responsable para todo lo relativo con el cómputo de totales. que permitiría mayor escalabilidad en el caso de querer añadir nuevos tipos de consumo de videos (p. ej. alquiler).

A este servicio, llamemosle PricingContext, que es quien tendría el getTotal, se le pasaría a su vez las distintas estrategias de pago por tipo de producto.

A su vez, tenemos otra cuestión. Por ahora tenemos definidos dos tipos de método de pago, el Download y el Streaming. Sin embargo, si pensamos de manera abstracta mejor pensar que pueden ser N realmente. Entonces partiendo de esta premisa, sería buena idea crear una interfaz/clase abstracta de la que heredarán las distintas estrategias que vayan surgiendo, pudiendo así cada una contener la lógica que les pertoque, pero partiendo de un nexo común.

Sería algo parecido a esto

```
class Media:
    type: string  // 'Streaming' or 'Download'
    price: float
    isPremium: bool
    additionalFee: float

# Clase base para las estrategias de precios
class PricingStrategy:
    function calculatePrice(media: Media): float:
        raise NotImplementedError

# Estrategia para precios de streaming
class StreamingPricingStrategy(PricingStrategy):
    function calculatePrice(media: Media): float:
        total = media.price
        if media.isPremium:
            total += media.additionalFee
        return total

# Estrategia para precios de descarga
class DownloadPricingStrategy(PricingStrategy):
    function calculatePrice(media: Media): float:
        total = media.price
        if media.isPremium:
            total += media.additionalFee
        return total
```

```
# Contexto que maneja las estrategias de precios
class PricingContext:
    strategies: map[string, PricingStrategy]

    function __init__():
        self.strategies = {}

    function addStrategy(mediaType: string, strategy: PricingStrategy):
        self.strategies[mediaType] = strategy

    function calculateTotal(mediaArray: array of Media): float:
        totalAmount = 0
        for media in mediaArray:
            strategy = self.strategies.get(media.type)
            if strategy:
                totalAmount += strategy.calculatePrice(media)
            else:
                raise ValueError("No strategy found for media type " + media.type)
        return totalAmount

# Clase de usuario registrada
class RegisteredUser:
    totalAmount: float

    function getTotal(mediaArray: array of Media, pricingContext: PricingContext): float:
        return pricingContext.calculateTotal(mediaArray)

# Configuración del contexto de precios con estrategias específicas
pricingContext = PricingContext()
pricingContext.addStrategy('Streaming', new StreamingPricingStrategy())
pricingContext.addStrategy('Download', new DownloadPricingStrategy())

# Uso del servicio para obtener los medios y cálculo del total
service = MediaService()
mediaArray = service.fetchMediaForUser(userId)
user = RegisteredUser()
user.totalAmount = user.getTotal(mediaArray, pricingContext)
```