



Integrantes

Agustín Perez Pesce

Mario Cristian Sánchez

Braian Troncoso

Santiago Mendoza

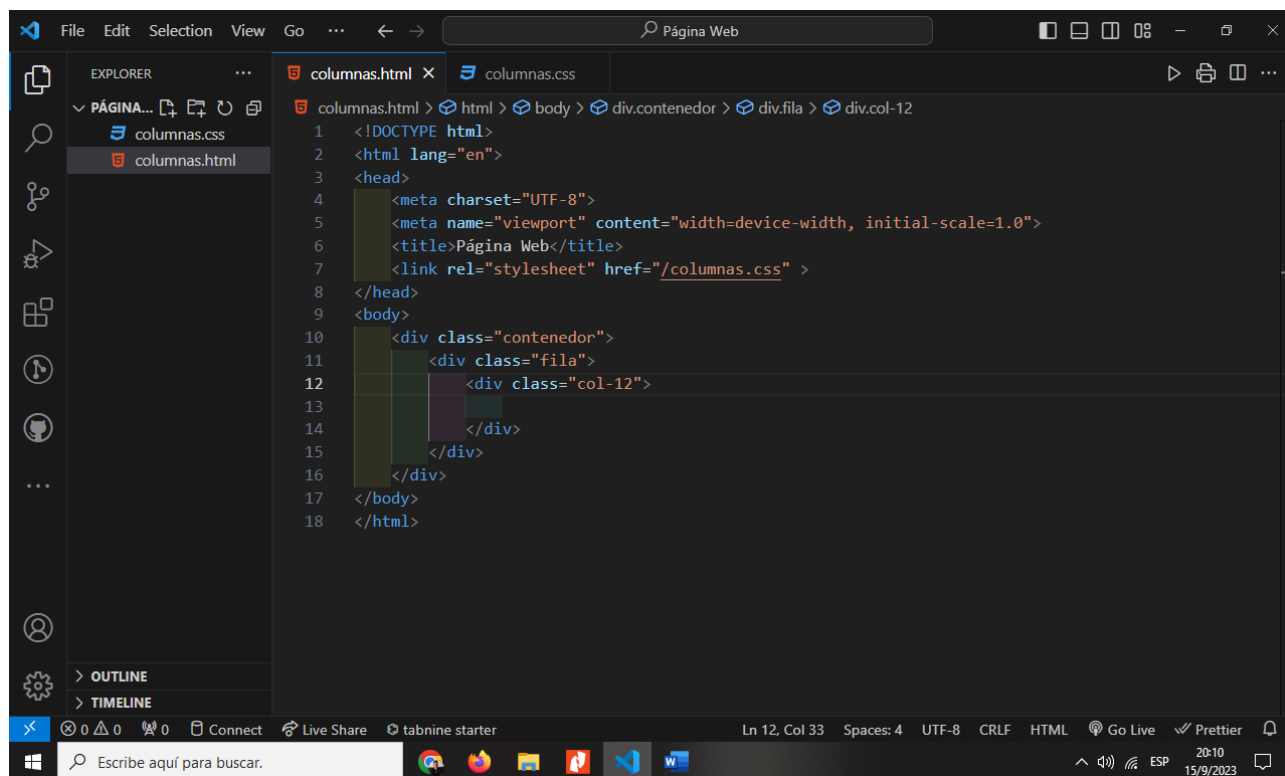
Franco Sebastián Genre

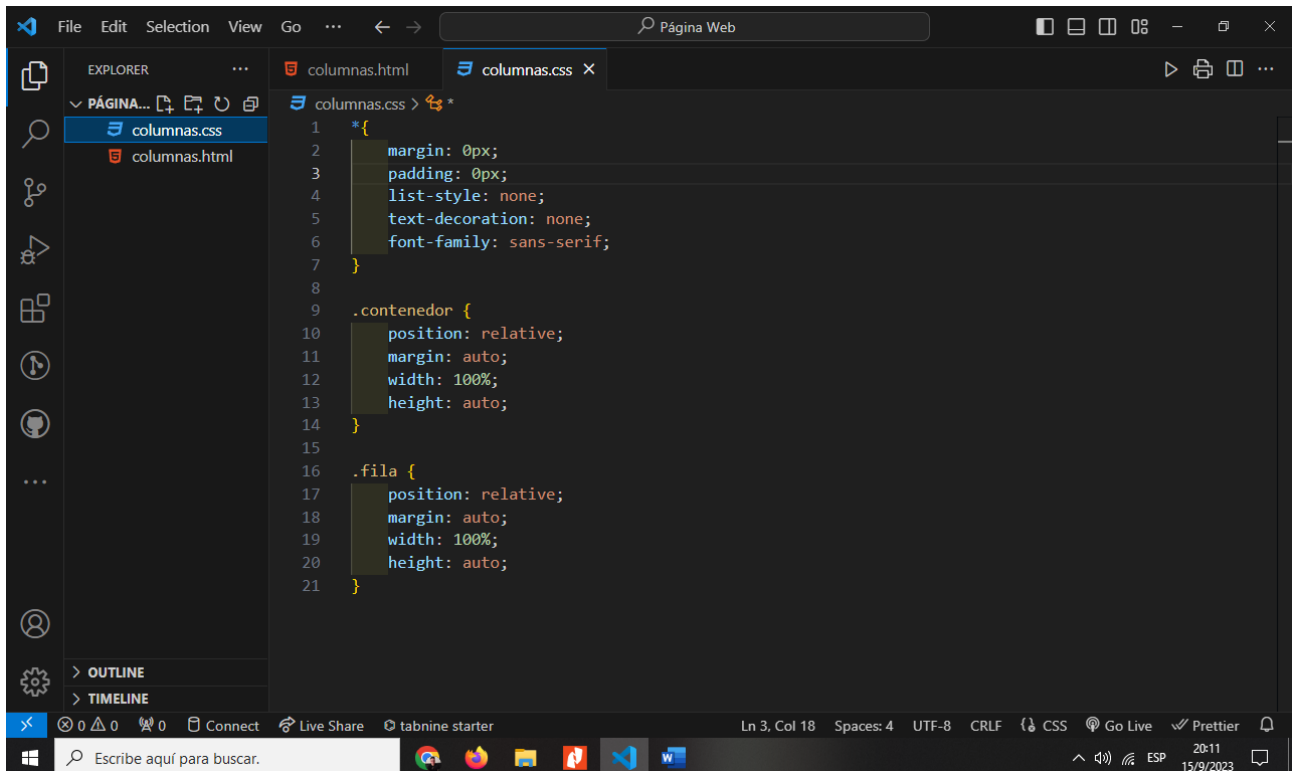
Sebastián Galván

Mariano Farias

Trabajo Práctico N° 3 – Base de Datos

Ω - Añadir etiquetas a la hoja columnas.html y enviar captura.





Ω Realizar un ejemplo de las dos propiedades SOLID vistas en clase.

Principio de Responsabilidad Única (SRP):

El Principio de Responsabilidad Única establece que una clase debe tener una sola razón para cambiar. Esto significa que una clase debe tener una única responsabilidad o función en el sistema, y cualquier cambio en esa responsabilidad debe conducir a cambios en esa clase y en ninguna otra.

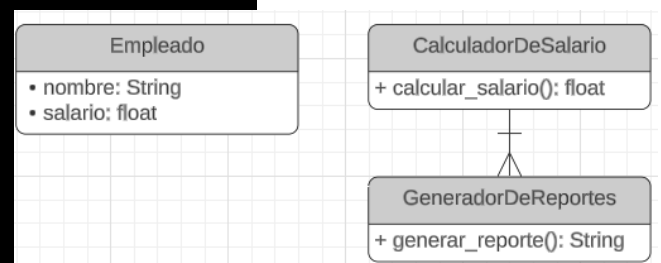
Ejemplo:

Supongamos que estamos desarrollando un sistema de gestión de empleados. En lugar de crear una única clase gigante que maneje todas las funcionalidades relacionadas con los empleados, aplicamos el SRP dividiendo las responsabilidades en diferentes clases:

```
class Empleado:
    def __init__(self, nombre, salario):
        self.nombre = nombre
        self.salario = salario

class CalculadoraDeSalario:
    @staticmethod
    def calcular_salario(empleado):
        # Cálculos complejos para determinar el salario
        return empleado.salario

class GeneradorDeReportes:
    @staticmethod
    def generar_reporte(empleado):
        # Generar un informe basado en los datos del empleado
        return f"Nombre: {empleado.nombre}, Salario: {empleado.salario}"
```





En este ejemplo, tenemos tres clases diferentes, cada una con una única responsabilidad: la clase Empleado almacena los datos del empleado, la clase CalculadoraDeSalario calcula el salario y la clase GeneradorDeReportes genera informes. Esto sigue el SRP, ya que cada clase tiene una razón para cambiar única.

Principio Abierto/Cerrado (OCP):

El Principio Abierto/Cerrado establece que las entidades (clases, módulos, funciones, etc.) deben estar abiertas para extenderse, pero cerradas para modificarse. Esto significa que debemos poder agregar nuevas funcionalidades o extensiones a un sistema sin modificar el código existente.

Ejemplo:

Supongamos que tenemos un sistema de procesamiento de pagos con un módulo de procesamiento de pagos existente. Aplicamos el OCP de la siguiente manera:

```
from abc import ABC, abstractmethod
```

```
class ProcesadorDePago(ABC):
```

```
    @abstractmethod
```

```
    def procesar_pago(self, monto):
```

```
        pass
```

```
class ProcesadorDePagoTarjeta(ProcesadorDePago):
```

```
    def procesar_pago(self, monto):
```

```
        # Lógica para procesar pagos con tarjeta de crédito
```

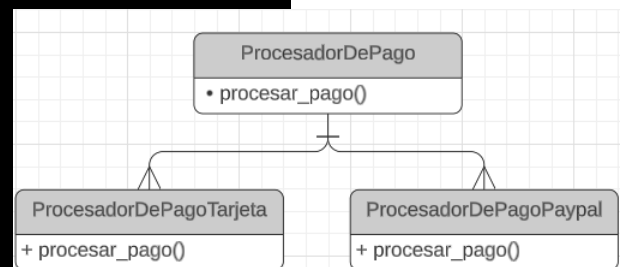
```
        print(f"Procesando pago con tarjeta: ${monto}")
```

```
class ProcesadorDePagoPayPal(ProcesadorDePago):
```

```
    def procesar_pago(self, monto):
```

```
        # Lógica para procesar pagos con PayPal
```

```
        print(f"Procesando pago con PayPal: ${monto}")
```



En este ejemplo, tenemos una clase abstracta ProcesadorDePago que define un método abstracto procesar_pago. Luego, creamos dos clases concretas (ProcesadorDePagoTarjeta y ProcesadorDePagoPayPal) que extienden la funcionalidad sin modificar la clase base. Esto sigue el OCP, ya que podemos agregar más procesadores de pago en el futuro sin modificar el código existente.