PROGRAMACIÓN AVANZADA

TRABAJO PRÁCTICO INTEGRADOR

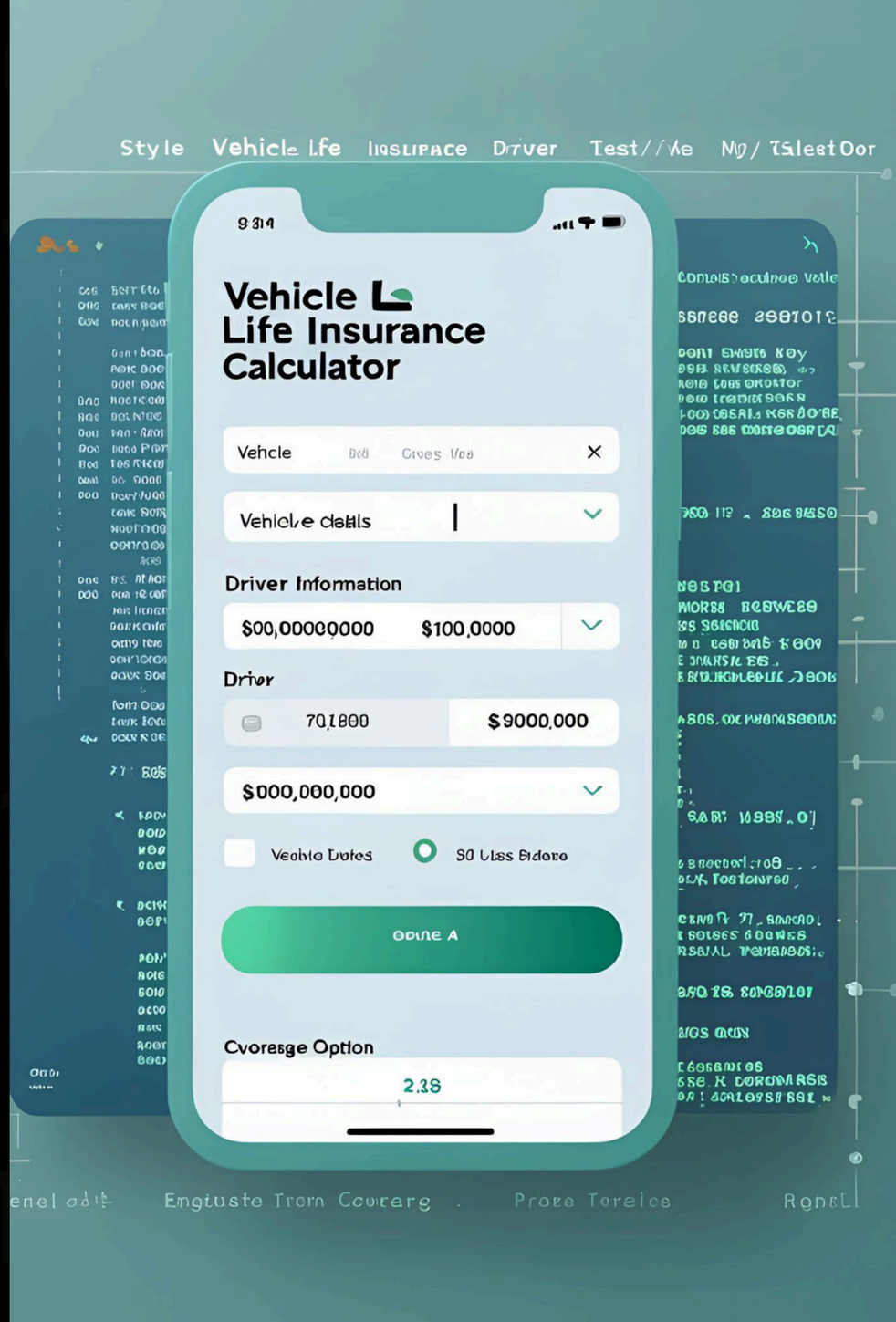
INTEGRANTES:
Sanchez Bajo, Mauro
Sosa Redchuk, Leandro Exequiel
Vera Espinola, Julieta

DOCENTE:
Piriz, Gianluca

Junio de 2025

CALCULADORA DE SEGUROS

SISTEMA DE COTIZACIÓN CON PROGRAMACIÓN ORIENTADA A OBJETOS



ESTRUCTURA DE LA PRESENTACIÓN

- MOTIVACIÓN Y OBJETIVOS
- METODOLOGÍA UTILIZADA
- DISEÑO DEL SISTEMA
- PATRONES DE DISEÑO IMPLEMENTADOS
- PRINCIPIOS SOLID APLICADOS
- DEMOSTRACIÓN DEL CÓDIGO
- RESULTADOS OBTENIDOS
- CONCLUSIONES Y PROYECCIONES

01/

MOTIVACIÓN Y OBJETIVOS

MOTIVACIÓN Y OBJETIVOS

¿POR QUÉ UNA CALCULADORA DE SEGUROS?

- **Problema real:** Dificultad para cotizar seguros entre múltiples opciones
- **Aplicación práctica:** Lógica de negocio variable y concreta
- **Flexibilidad:** Diferentes tipos de seguros con reglas específicas
- **Extensibilidad:** Posibilidad de agregar descuentos y nuevos tipos

OBJETIVOS DEL PROYECTO:

✓ Consolidar conocimientos de POO
✓ Aplicar patrones de diseño en casos reales
✓ Desarrollar sistema mantenible y escalable

02/

METODOLOGÍA UTILIZADA

METODOLOGÍA

HERRAMIENTAS Y TECNOLOGÍAS

- **Lenguaje:** Python 3.x
- **Control de versiones:** GitHub
- **Testing:** Terminal y consola
- **Enfoque:** POO con patrones de diseño

PROCESO DE DESARROLLO:

1. Análisis del problema
2. Diseño de la arquitectura
3. Implementación de patrones
4. Testing y validación
5. Documentación

03/

DISEÑO DEL SISTEMA

ARQUITECTURA DEL SISTEMA

COMPONENTES PRINCIPALES

- **Estrategias de cálculo:** Algoritmos específicos por tipo de seguro
- **Seguro concreto:** Representa un seguro individual
- **Decoradores:** Modificaciones dinámicas (descuentos)
- **Fábrica:** Creación controlada de estrategias

FLUJO DE DATOS:

Usuario ➤ Datos ➤ Factory ➤ Estrategia ➤ Cálculo ➤ Decorador ➤ Resultado

JERARQUÍA DE CLASES

Interfaces/Abstractas:

- SeguroBase (ABC)
- EstrategiaCalculo (ABC)

Implementaciones:

- SeguroConcreto → SeguroBase
- CalculoSeguroAuto → EstrategiaCalculo
- CalculoSeguroVida → EstrategiaCalculo

Patrones:

- DescuentoClienteNuevo (Decorator)
- SeguroFactory (Factory Method)

04/

PATRONES DE DISEÑO IMPLEMENTADOS

PATRÓN STRATEGY

Encapsula algoritmos de cálculo intercambiables

Implementación:

```
class CalculoSeguroAuto(EstrategiaCalculo):
    def calcular(self, datos: dict) -> float:
        anio_actual = 2025
        tarifa_base = 10000
        antiguedad = anio_actual - datos.get("anio_auto", anio_actual)
        return tarifa_base + antiguedad * 500
```

Beneficios:

- Algoritmos intercambiables
- Fácil agregar nuevos tipos
- Separación de responsabilidades

PATRÓN DECORATOR

Modifica comportamiento dinámicamente

Implementación:

```
class DescuentoClienteNuevo(SeguroBase):
    def __init__(self, seguro: SeguroBase):
        self.seguro = seguro

    def calcular(self) -> float:
        return self.seguro.calcular() * 0.9 # 10% descuento
```

Ventajas:

- Sin modificar código original
- Funcionalidades apilables
- Flexibilidad en tiempo de ejecución

PATRÓN FACTORY METHOD

Creación controlada de objetos

Implementación:

```
class SeguroFactory:
    @staticmethod
    def crear_seguro(tipo: str) -> EstrategiaCalculo:
        if tipo == "auto":
            return CalculoSeguroAuto()
        elif tipo == "vida":
            return CalculoSeguroVida()
        else:  raise ValueError("Tipo de seguro no reconocido")
```

Beneficios:

- Desacoplamiento
- Centraliza lógica de creación
- Facilita mantenimiento

05/

PRINCIPIOS SOLID APLICADOS

PRINCIPIOS SOLID

S - Single Responsibility: Cada clase tiene una única responsabilidad

O - Open/Closed: Abierto a extensión, cerrado a modificación

L - Liskov Substitution: Subtipos reemplazables

I - Interface Segregation: Interfaces específicas y pequeñas

D - Dependency Inversion: Depende de abstracciones, no concreciones

Aplicación en nuestro código:

✅ Cada clase cumple una función específica

✅ Nuevos seguros sin modificar existentes

✅ Polimorfismo efectivo

✅ Interfaces mínimas y claras

PILARES DE POO APLICADOS

Encapsulamiento
- Datos protegidos dentro de objetos
- Métodos públicos controlados

Herencia
- Clases concretas extienden interfaces base
- Reutilización de comportamiento

Abstracción
- Interfaces generales (SeguroBase, EstrategiaCalculo)
- Trabajo genérico sin conocer implementación

Polimorfismo
- Intercambiabilidad de implementaciones
- Mismo método, diferentes comportamientos

06/

DEMOSTRACIÓN DEL CÓDIGO

DEMOSTRACIÓN DEL CÓDIGO

Ejemplo de uso completo:

```
# Datos del cliente
datos_auto = {"anio_auto": 2020}
datos_vida = {"edad": 40}

# Crear estrategias desde la fábrica
estrategia_auto = SeguroFactory.crear_seguro("auto")
estrategia_vida = SeguroFactory.crear_seguro("vida")

# Crear seguros
seguro_auto = SeguroConcreto(estrategia_auto, datos_auto)
seguro_vida = SeguroConcreto(estrategia_vida, datos_vida)

# Aplicar decorador de descuento
seguro_auto_descuento = DescuentoClienteNuevo(seguro_auto)

# Resultados
print("Seguro de auto:", seguro_auto.calcular())
print("Seguro de auto con descuento:", seguro_auto_descuento.calcular())
print("Seguro de vida:", seguro_vida.calcular())
```

07/

RESULTADOS OBTENIDOS

RESULTADOS OBTENIDOS

Objetivos cumplidos:

- Sistema flexible y extensible
- Aplicación correcta de patrones de diseño
- Código mantenible y escalable
- Separación clara de responsabilidades

Métricas del proyecto:

- 7 clases implementadas

(EstrategiaCalculo (Clase abstracta); SeguroBase (Clase abstracta); CalculoSeguroAuto (Implementación concreta); CalculoSeguroVida (Implementación concreta); SeguroConcreto (Implementación concreta); DescuentoClienteNuevo (Decorador); SeguroFactory (Fábrica)

- 3 patrones de diseño aplicados

(Strategy Pattern, Decorator Pattern, Factory Method Pattern)

- 5 principios SOLID respetados

(S - Single Responsibility, O - Open/Closed, L - Liskov Substitution, I - Interface Segregation, D - Dependency Inversion)

- 4 pilares POO aplicados

(Encapsulamiento, Abstracción, Herencia y Polimorfismo)

APRENDIZAJES Y DIFICULTADES

Principales aprendizajes:

- Patrones de diseño como herramientas reales
- Importancia del diseño pensando en cambios futuros
- Valor de separar responsabilidades
- Trabajo colaborativo efectivo

Dificultades enfrentadas:

- Entender cuándo y por qué usar cada patrón
- Aplicar teoría a casos prácticos
- Decisiones de diseño grupales

Soluciones aplicadas:

- Investigación de ejemplos
- Revisión de material académico
- Pruebas iterativas

08/

CONCLUSIONES Y PROYECCIONES

PROYECCIONES FUTURAS

Extensiones posibles:

- Nuevos tipos de seguros (hogar)
- Sistema de registro de usuarios
- Integración con base de datos
- Generación de reportes (PDF/CSV)
- Conexión con plataformas de pago

Beneficios del diseño actual:

- Desacoplamiento: Fácil modificación
- Extensibilidad: Agregar funciones sin romper código
- Reutilización: Componentes combinable

CONCLUSIONES

Logros técnicos:

- Integración práctica de conceptos POO
- Implementación exitosa de 3 patrones de diseño
- Sistema preparado para el cambio y crecimiento

Valor del proyecto:

- Solución a problema real
- Código mantenible y profesional
- Base sólida para expansiones futuras

Impacto en el aprendizaje:

- Consolidación de conocimientos teóricos
- Experiencia en diseño de software
- Comprensión del valor del buen diseño

MUCHAS GRACIAS
POR SU ATENCION!