PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

PROGRAMACIÓN 3
1ra práctica (tipo b)
(Segundo Semestre 2025)

Indicaciones Generales:

- Duración 1h 50 minutos (Inicio: 8:00 a.m. - Fin: 9:50 a.m.) (Subida y verificación de archivos en PAIDEIA: 9:50 a.m. a 9:55 a.m.) (Salida del Laboratorio: 9:55 a.m. a 10:00 a.m.)
- Se les recuerda que, de acuerdo con el reglamento disciplinario de nuestra institución, constituye una falta grave copiar del trabajo realizado por otro estudiante o cometer plagio para el desarrollo de esta práctica.
- Para el desarrollo de toda la práctica debe utilizar el sistema operativo Windows y el .NET Framework 4.8.
- Para el desarrollo de las preguntas debe usar Microsoft Visual Studio .NET.
- Está permitido el uso de Internet (únicamente para consultar páginas oficiales de Oracle). No obstante, está prohibida toda forma de comunicación con otros estudiantes o terceros.
- PUEDE UTILIZAR MATERIAL DE CONSULTA. Antes de comenzar el laboratorio, descargue todos los proyectos, apuntes, diapositivas que utilizará.
- Se considerará en la calificación el uso de buenas prácticas de programación (aquellas vistas en clase).

Puntaje total: 20 puntos

Cuestionario

- La finalidad principal de este laboratorio es la de reforzar los conceptos contenidos en las clases 01, 02 y 03: Programación Orientada a Objetos y manejo de librerías.
- Al finalizar la práctica, comprima la carpeta dada en las indicaciones iniciales empleando el programa Zip que viene por defecto en el Windows, no se aceptarán los trabajos compactados con otros programas como RAR, WinRAR, 7zip o similares.

Descripción del caso

Una aseguradora especializada en el rubro automotriz ha identificado la necesidad de modernizar su sistema de cotizaciones, con el fin de brindar mayor **precisión, rapidez y flexibilidad** en la oferta de seguros vehiculares a sus clientes. Actualmente, la compañía maneja un catálogo estático de seguros que no se adapta fácilmente a las diferentes combinaciones de características de los vehículos y a los perfiles de riesgo asociados, lo que limita su competitividad en el mercado.

El nuevo sistema deberá estar orientado al cálculo dinámico de pólizas, partiendo de información básica pero crucial de cada vehículo, como su **marca** y **antigüedad**. Estos atributos serán determinantes para aplicar diferentes reglas de negocio, que impactarán directamente en la cotización final:

- La **marca** puede estar asociada a distintos niveles de costo de reparación o disponibilidad de repuestos, lo que influye en el riesgo asumido por la aseguradora.
- La **antigüedad** del vehículo será un factor crítico para calcular depreciación, probabilidad de fallas mecánicas o vulnerabilidad frente a accidentes.

Para lograr flexibilidad, el sistema deberá estar diseñado con una **jerarquía de clases y herencia** que permita modelar distintos tipos de seguros vehiculares (por ejemplo: seguro básico contra terceros, seguro contra todo riesgo, seguro con asistencia en carretera, etc.). De esta manera, cada tipo de seguro podrá especializar sus reglas de cálculo y coberturas, sin necesidad de reescribir la lógica general de cotización.

Además, el sistema debe ser capaz de adaptarse a cambios futuros en las políticas de la aseguradora, ya que el mercado es dinámico y requiere que las condiciones de cotización puedan evolucionar con rapidez. Esto implica que el diseño debe contemplar **extensibilidad**, de modo que puedan agregarse nuevos tipos de seguros o nuevas variables de cálculo sin necesidad de modificar la estructura central.

En resumen, la aseguradora requiere un sistema que:

- Permita ingresar la marca y antigüedad del vehículo para cotizar.
- Genere diferentes opciones de seguros vehiculares, cada una con reglas propias y precios diferenciados.
- Sea flexible y extensible, facilitando la incorporación de nuevas modalidades de seguros en el futuro.
- Garantice precisión y rapidez en la generación de las cotizaciones para mejorar la experiencia del cliente y la competitividad de la empresa.

El programa deberá ser capaz de:

1. Definir, organizar y cotizar las distintas coberturas y seguros vehiculares disponibles, soportando una jerarquía flexible de objetos que aproveche los principios de encapsulamiento, herencia y polimorfismo para extender o modificar fácilmente los productos aseguradores.
2. Generar cotizaciones personalizadas en función de las características del vehículo (marca, antigüedad y tipo de cobertura), permitiendo ajustes dinámicos según las reglas de negocio.
3. Aplicar el patrón estructural Decorador para gestionar y combinar de manera flexible los distintos tipos de coberturas, evitando una explosión de subclases y facilitando la extensión del sistema.

Pregunta 01 (12 puntos):

Implemente una jerarquía de clases que define los diferentes tipos de seguros disponibles (Seguro Básico, Seguro Bronce: Básico + Cobertura contra Robos, Seguro Plata: Bronce + Asistencia Vial y Seguro Oro: Plata + Cobertura contra desastres naturales) como se muestra en el siguiente diagrama de clases.

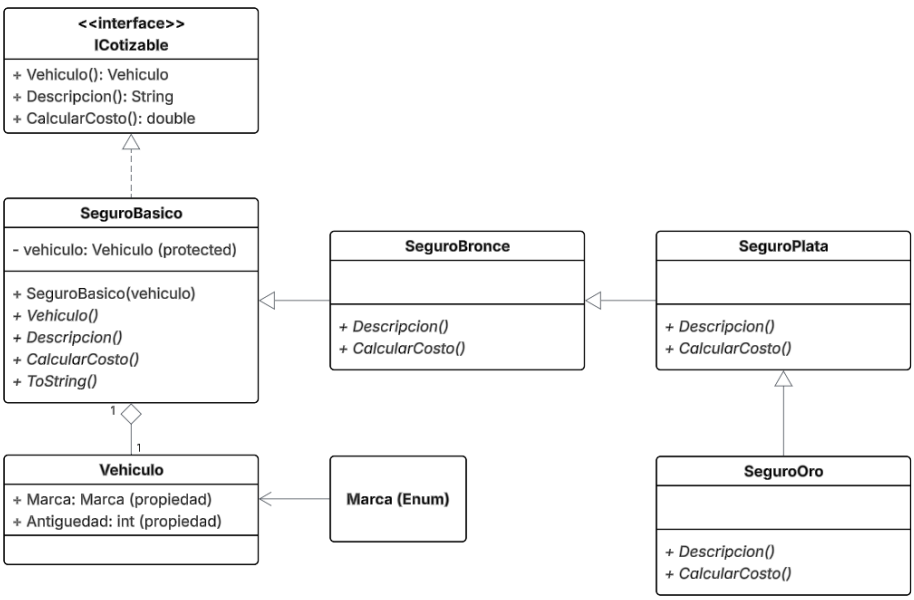

Figura 1 - Diagrama de clases Pregunta 01

La función Main deberá contener el siguiente código fuente.

```
using System;
using Lab01.Pregunta01;

namespace Lab01 {
   public class Program {
      public static void Main(string[] args) {
         Vehiculo vehiculo = new Vehiculo {
            Antiguedad = 10,
            Marca = Marca.Toyota
         };

         Console.WriteLine("===== SEGUROS PREDEFINIDOS =====");

         ICotizable seguro = new SeguroBasico(vehiculo);
         Console.WriteLine(seguro);

         seguro = new SeguroBronce(vehiculo);
         Console.WriteLine(seguro);

         seguro = new SeguroPlata(vehiculo);
         Console.WriteLine(seguro);

         seguro = new SeguroOro(vehiculo);
         Console.WriteLine(seguro);
      }
   }
}
```

La pregunta consta de tres partes que deben resolverse de manera secuencial.

Parte 01 (2 puntos)

Abra Visual Studio .NET y cree un nuevo proyecto de tipo *Console App (.NET Framework)*. Asigne al proyecto el nombre Lab01 como se muestra en la Figura 2.

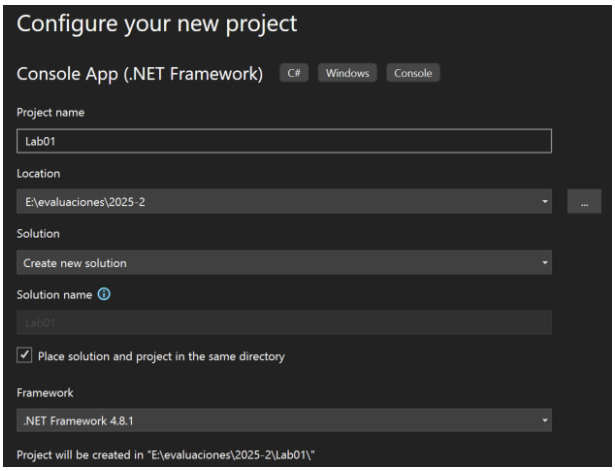

Figura 2 - Creación del Proyecto

Agregue un folder llamado Pregunta01 y cree las siguientes clases, interfaces y enumeraciones dentro.

	Nombre del Archivo	Tipo
1	ICotizable.cs	Interfaz
2	Marca.cs	Enumeración

3	SeguroBasico.cs	Clase
4	SeguroBronce.cs	Clase
5	SeguroOro.cs	Clase
6	SeguroPlata.cs	Clase
7	Vehiculo.cs	Clase

Tabla 1 - Archivos Pregunta 01

Al finalizar, el proyecto deberá contener el folder y los archivos como en la Figura 3.

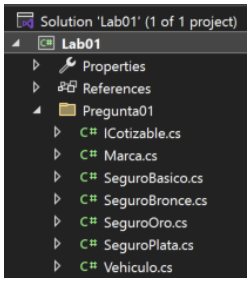

Figura 3 - Estructura de Archivos Pregunta 01

Parte 02 (2 puntos)

En el archivo Marca.cs defina la enumeración **Marca** con los siguientes elementos.

- Toyota
- Nissan
- Hyundai
- Audi
- Mercedez
- BMW

En el archivo Vehiculo.cs defina la clase **Vehiculo** con las siguientes propiedades, se recomienda usar la implementación automática de propiedades e.j., *public string Nombre { get; set; }*

	Miembro	Tipo
1	Marca	Marca (Enum)
2	Antiguedad	int

Tabla 2 - Clase Vehículo

En el archivo ICotizable.cs defina la interfaz ICotizable con los siguientes métodos.

	Método	Tipo de Retorno	Parámetros
1	Vehiculo	Vehiculo	NA
2	Descripcion	string	NA
3	CalcularCosto	double	NA

Tabla 3 - Interfaz ICotizable

Parte 03 (8 puntos):

En el archivo SeguroBasico.cs defina la clase **SeguroBasico**, esta clase debe implementar la interfaz ICotizable y definir los siguientes atributos y métodos.

Atributos			
	Nombre	**Tipo**	**Opciones**
1	vehiculo	Vehiculo	Acceso: protected
Métodos			
	Nombre	**Tipo de Retorno**	**Opciones**
1	Constructor	NA	Tipo: Propiamente dicho Parámetros: vehiculo (Vehiculo)
2	CalcularCosto	double	Parámetros: NA Acceso: public Enlace: virtual

3	Vehiculo	Vehiculo	Parámetros: NA Acceso: public Enlace: virtual
4	Descripcion	string	Parámetros: NA Acceso: public Enlace: virtual
5	ToString	string	Parámetros: NA Acceso: public Enlace: override

Tabla 4 - Clase Seguro Básico

El seguro básico tiene las siguientes reglas para el calcular su costo.

- Costo base S/ 500.
- Si la antigüedad del vehículo es mayor a 10 años el costo incrementa en S/ 200.00 adicionales.
- Si la antigüedad del vehículo es mayor a 5 años el costo incrementa en S/ 100.00 adicionales.
- Si la marca del vehículo es una marca premium (Audi, BMW, Mercedez) el costo incrementa en S/ 100.00 adicionales.

La descripción del seguro básico es “Seguro básico vehicular”.
El método ToString debe retornar una cadena que muestra el Precio y la Descripción.

```
public override string ToString() {
   return $"Precio: {this.CalcularCosto():F2}, Descripción: {this.Descripcion()}";
}
```

En el archivo SeguroBronce.cs defina la clase **SeguroBronce** la cual hereda de la clase **SeguroBasico** con los siguientes atributos y métodos.

Métodos			
	Nombre	**Tipo de Retorno**	**Opciones**
1	Constructor	NA	Tipo: Propiamente dicho Parámetros: vehiculo (Vehiculo) Debe llamar al constructor de la clase base
2	CalcularCosto	double	Parámetros: NA Acceso: public Enlace: override
3	Descripcion	string	Parámetros: NA Acceso: public Enlace: override

Tabla 5 - Clase SeguroBronce

El seguro bronce tiene las siguientes reglas para el calcular su costo.

- Costo del seguro básico + S/ 100.00.
- Si la marca del vehículo es Toyota el costo incrementa en S/ 100.00 adicionales.

La descripción del seguro bronce es la descripción del seguro básico más “cobertura contra robos”.

```
public override string Descripcion() {
   return base.Descripcion() + " + cobertura contra robos";
}
```

En el archivo SeguroPlata.cs defina la clase **SeguroPlata** la cual hereda de la clase **SeguroBronce** con los siguientes atributos y métodos.

Métodos			
	Nombre	**Tipo de Retorno**	**Opciones**
1	Constructor	NA	Tipo: Propiamente dicho Parámetros: vehiculo (Vehiculo) Debe llamar al constructor de la clase base

2	CalcularCosto	double	Parámetros: NA Acceso: public Enlace: override
3	Descripcion	string	Parámetros: NA Acceso: public Enlace: override

Tabla 6 - Tabla Seguro Plata

El seguro plata tiene las siguientes reglas para el calcular su costo.

- Costo del seguro bronce + S/ 100.00.

La descripción del seguro plata es la descripción del seguro bronce más “asistencia vial”.

```
public override string Descripcion() {
   return base.Descripcion() + " + asistencia vial";
}
```

En el archivo SeguroOro.cs defina la clase **SeguroOro** la cual hereda de la clase **SeguroPlata** con los siguientes atributos y métodos.

Métodos			
	Nombre	**Tipo de Retorno**	**Opciones**
1	Constructor	NA	Tipo: Propiamente dicho Parámetros: vehiculo (Vehiculo) Debe llamar al constructor de la clase base
2	CalcularCosto	double	Parámetros: NA Acceso: public Enlace: override
3	Descripcion	string	Parámetros: NA Acceso: public Enlace: override

Tabla 7 - Clase SeguroOro

El seguro plata tiene las siguientes reglas para el calcular su costo.

- Costo del seguro plata + S/ 300.00.

La descripción del seguro oro es la descripción del seguro plata más “cobertura contra desastres naturales”.

```
public override string Descripcion() {
   return base.Descripcion() + " + cobertura contra desastres naturales";
}
```

Pregunta 02 (8 puntos):

La empresa aseguradora requiere mayor flexibilidad en la cotización de seguros vehiculares, ya que algunos clientes solicitan, por ejemplo, el seguro básico con solo asistencia vial, el seguro básico con únicamente cobertura contra desastres naturales u otras combinaciones. El uso de la jerarquía de clases definida en la Pregunta 01 implicaría la creación de un gran número de clases para manejar todas las combinaciones posibles, considerando además que continuamente se incorporan nuevos tipos de cobertura.

Para ello se aplicará el patrón Decorador, que permitirá gestionar de manera eficiente los distintos tipos de cobertura.

El patrón **Decorador** es un patrón estructural que permite añadir responsabilidades adicionales a un objeto de forma dinámica, sin modificar su estructura ni la clase original. Se basa en envolver el objeto dentro de otro objeto decorador que implementa la misma interfaz y delega las operaciones al objeto envuelto, añadiendo o extendiendo comportamientos según sea necesario. De esta manera, se favorece

la composición sobre la herencia, ofreciendo mayor flexibilidad y evitando la proliferación de subclases para cada combinación de funcionalidades.

Se definirá una clase abstracta SeguroDecorador, a partir de la cual se crearán clases concretas que representen cada tipo de cobertura disponible (CoberturaContraRobos, CoberturaAsistenciaVial, CoberturaContraDesastresNaturales). Como se muestra en el siguiente diagrama de clases.

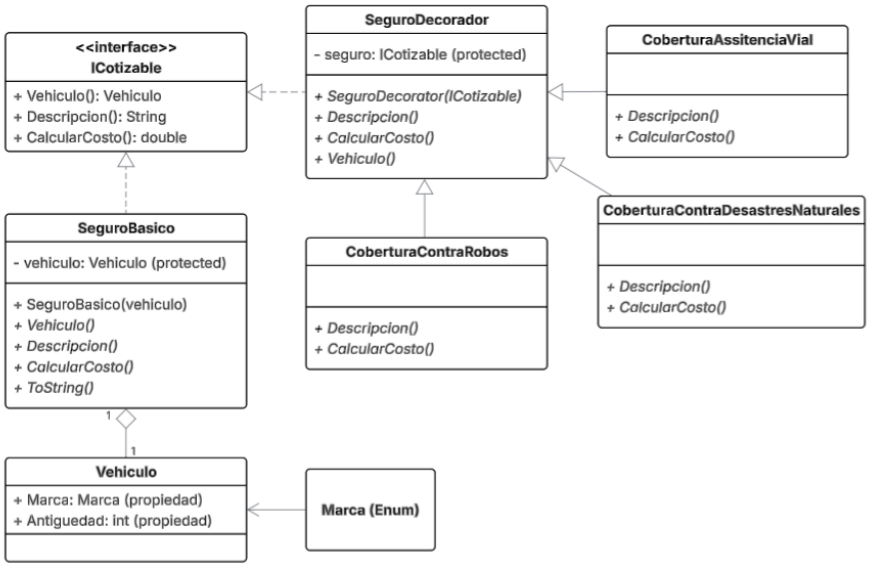

Figura 4 - Diagrama de Clases Pregunta02

La función Main deberá contener el siguiente código fuente.

```
using System;
using Lab01.Pregunta01;
using Lab01.Pregunta02;

namespace Lab01 {
   public class Program {
      public static void Main(string[] args) {
         Vehiculo vehiculo = new Vehiculo {
            Antiguedad = 10,
            Marca = Marca.Toyota
         };

         Console.WriteLine("===== SEGUROS PREDEFINIDOS =====");

         ICotizable seguro = new SeguroBasico(vehiculo);
         Console.WriteLine(seguro);

         seguro = new SeguroBronce(vehiculo);
         Console.WriteLine(seguro);

         seguro = new SeguroPlata(vehiculo);
         Console.WriteLine(seguro);

         seguro = new SeguroOro(vehiculo);
         Console.WriteLine(seguro);

         /* Implementación utilizando el Patrón Decorador
          * para extender funcionalidades de manera flexible */
         Console.WriteLine("\n===== SEGUROS DECORADOS (Patrón Decorador) =====");

         seguro = new SeguroBasico(vehiculo);
         Console.WriteLine(seguro);

         seguro = new SeguroBasico(vehiculo)
```

```
            .ConCoberturaContraRobos();
        Console.WriteLine(seguro);

        seguro = new SeguroBasico(vehiculo)
            .ConCoberturaContraRobos()
            .ConAsistenciaVial();
        Console.WriteLine(seguro);

        seguro = new SeguroBasico(vehiculo)
            .ConCoberturaContraRobos()
            .ConAsistenciaVial()
            .ConCoberturaContraDesastresNaturales();
        Console.WriteLine(seguro);
    }
  }
}
```

La pregunta consta de dos partes que deben resolverse de manera secuencial.

Parte 01 (2 puntos):

Agregue un folder llamado Pregunta02 y cree las siguientes clases dentro.

	Nombre del Archivo	Tipo
1	CoberturaAsistenciaVial.cs	Clase
2	CoberturaContraDesastresNaturales.cs	Clase
3	CoberturaContraRobos.cs	Clase
4	SeguroDecorador.cs	Clase
5	Extensiones.cs	Clase

Tabla 8 - Archivos Pregunta 02

Al finalizar, el proyecto deberá contener el folder y los archivos como en la Figura 5.

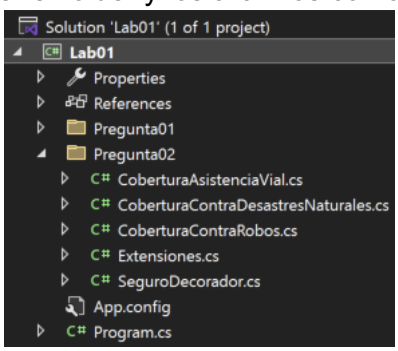

Figura 5 – Estructura de Archivos Pregunta 02

El archivo Extensiones.cs deberá tener este contenido.

```
using Lab01.Pregunta01;

namespace Lab01.Pregunta02 {
  public static class ExtensionesDeICotizable {
    public static ICotizable ConCoberturaContraRobos(this ICotizable cotizable) {
      return new CoberturaContraRobos(cotizable);
    }

    public static ICotizable ConAsistenciaVial(this ICotizable cotizable) {
      return new CoberturaAsistenciaVial(cotizable);
    }

    public static ICotizable ConCoberturaContraDesastresNaturales(this ICotizable cotizable) {
      return new CoberturaContraDesastresNaturales(cotizable);
    }
  }
}
```

Parte 02 (6 puntos):

En el archivo SeguroDecorador.cs defina la clase **SeguroDecorador**, esta clase debe implementar la interfaz ICotizable y definir los siguientes atributos y métodos.

Atributos			
	Nombre	**Tipo**	**Opciones**
1	seguro	ICotizable	Acceso: protected
Métodos			
	Nombre	**Tipo de Retorno**	**Opciones**
1	Constructor	NA	Tipo: Propiamente dicho Parámetros: seguro (ICotizable)
2	CalcularCosto	double	Parámetros: NA Acceso: public Enlace: virtual
3	Vehiculo	Vehiculo	Parámetros: NA Acceso: public Enlace: virtual
4	Descripcion	string	Parámetros: NA Acceso: public Enlace: virtual
5	ToString	string	Parámetros: NA Acceso: public Enlace: override

Tabla 9 - Clase Seguro Decorador

El método CalcularCosto debe llamar al método CalcularCosto del atributo seguro.

```
public virtual double CalcularCosto() {
   return this.seguro.CalcularCosto();
}
```

El método Descripcion debe llamar al método Descripcion del atributo seguro.

```
public virtual string Descripcion() {
   return this.seguro.Descripcion();
}
```

El método Vehiculo debe llamar al método Vehiculo del atributo seguro.

```
public Vehiculo Vehiculo() {
   return this.seguro.Vehiculo();
}
```

El método ToString debe tener la siguiente implementación

```
public override string ToString() {
   return $"Precio: {this.CalcularCosto():F2}, Descripción: {this.Descripcion()}";
}
```

En el archivo CoberturaContraRobos.cs defina la clase CoberturaContraRobos, esta clase debe heredar de la clase SeguroDecorador y definir los siguientes atributos y métodos.

Métodos			
	Nombre	**Tipo de Retorno**	**Opciones**
1	Constructor	NA	Tipo: Propiamente dicho Parámetros: seguro (ICotizable) Debe llamar al constructor de la clase base

2	CalcularCosto	double	Parámetros: NA Acceso: public Enlace: override
3	Descripcion	string	Parámetros: NA Acceso: public Enlace: override

Tabla 10 - Clase Cobertura Contra Robos

El método CalcularCosto debe llamar al método CalcularCosto del atributo seguro y agregar S/ 100.00 y en el caso que la marca del vehiculo sea Toyota agregar S/ 100.00 adicionales.

```
public override double CalcularCosto() {
   double costo = 100.00;

   if (this.seguro.Vehiculo().Marca == Marca.Toyota) {
      costo += 100.00;
   }

   return this.seguro.CalcularCosto() + costo;
}
```

El método Descripcion debe llamar al método Descripcion del atributo seguro y agregar “cobertura contra robos”.

```
public override string Descripcion() {
   return this.seguro.Descripcion() + ", cobertura contra robos";

}
```

En el archivo CoberturaAsistenciaVial.cs defina la clase CoberturaAsistenciaVial, esta clase debe heredar de la clase SeguroDecorador y definir los siguientes atributos y métodos.

Métodos			
	Nombre	**Tipo de Retorno**	**Opciones**
1	Constructor	NA	Tipo: Propiamente dicho Parámetros: seguro (ICotizable) Debe llamar al constructor de la clase base
2	CalcularCosto	double	Parámetros: NA Acceso: public Enlace: override
3	Descripcion	string	Parámetros: NA Acceso: public Enlace: override

Tabla 11 - Clase Cobertura Asistencia Vial

El método CalcularCosto debe llamar al método CalcularCosto del atributo seguro y agregar S/ 100.00.

```
public override double CalcularCosto() {
   return this.seguro.CalcularCosto() + 100.00;
}
```

El método Descripcion debe llamar al método Descripcion del atributo seguro y agregar “asistencia vial”.

```
public override string Descripcion() {
   return this.seguro.Descripcion() + ", asistencia vial";

}
```

En el archivo CoberturaContraDesastresNaturales.cs defina la clase CoberturaContraDesastresNaturales, esta clase debe heredar de la clase SeguroDecorador y definir los siguientes atributos y métodos.

Métodos			
	Nombre	**Tipo de Retorno**	**Opciones**
1	Constructor	NA	Tipo: Propiamente dicho Parámetros: seguro (ICotizable) Debe llamar al constructor de la clase base
2	CalcularCosto	double	Parámetros: NA Acceso: public Enlace: override
3	Descripcion	string	Parámetros: NA Acceso: public Enlace: override

Tabla 12 - Clase Cobertura Contra Desastres Naturales

El método CalcularCosto debe llamar al método CalcularCosto del atributo seguro y agregar S/ 300.00.

```
public override double CalcularCosto() {
   return this.seguro.CalcularCosto() + 300.00;
}
```

El método Descripcion debe llamar al método Descripcion del atributo seguro y agregar “cobertura contra desastres naturales”.

```
public override string Descripcion() {
   return this.seguro.Descripcion() + ", cobertura contra desastres naturales";

}
```

Profesores del curso: Freddy Paz
Eric Huiza
Andrés Melgar

Pando, 27 de agosto de 2025