PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

PROGRAMACIÓN 3
Examen 1
(Segundo semestre 2025)

Indicaciones generales:

- **Duración:** 3 horas.
- **Materiales o equipos a utilizar:**
  La sección teórica se realizará sin material de apoyo.
  En la sección práctica podrá utilizar material de apoyo: diapositivas, ejemplos de clase, grabaciones de clase, código fuente y documentación oficial de Microsoft u Oracle.
- No está permitido el uso de ningún material o equipo electrónico adicional al indicado.
- **La presentación, la ortografía y la gramática influirán en la calificación.**

Puntaje total: 20 puntos

PARTE TEÓRICA: (5 puntos)

Preguntas Teóricas

Responda en PAIDEIA dos preguntas teóricas de las tres que han sido propuestas. (2.5 puntos cada una)

Pregunta 01

La arquitectura de los Formularios Web de ASP.NET implementa una clara separación entre el markup, encargado de definir la estructura, el contenido y la presentación visual de la aplicación, y el code-behind, que contiene el manejo de eventos y la lógica del lado del servidor.
Esta separación constituye uno de los pilares fundamentales del desarrollo en ASP.NET, ya que establece una relación definida entre la capa visual y la lógica de control.

1. Liste y explique que archivos están involucrados en la implementación de un Formulario Web en ASP .NET.
2. En su opinión, cuales fueron los objetivos del equipo de diseño de ASP .NET que llevaron a la separación entre markup y code-behind.
3. Dado el siguiente fragmento de markup:

   <asp:Button ID="btnSubmit" runat="server" Text="Enviar" OnClick="btnSubmit_Click" />

   Indique la estructura del método correspondiente en el archivo code-behind. No es necesario incluir la implementación interna del método.

Pregunta 02

Suponga que debe desarrollar un sitio web para una biblioteca pública. Algunas secciones son informativas (horarios, ubicación), otras permiten interacción (búsqueda de libros, reservas). ¿Qué partes implementaría como páginas estáticas y cuáles como dinámicas? Justifique su decisión en función de mantenimiento, rendimiento y experiencia de usuario.

Pregunta 03

Explique por qué crear múltiples instancias del objeto **DBManager** puede afectar el rendimiento y la estabilidad del sistema. Justifique cómo el uso del patrón **Singleton** contribuye a solucionar este problema dentro de una arquitectura orientada a objetos. Asimismo, explique claramente el motivo por el cual el constructo es privado y la variable es declarada dentro de la clase como privada y estática.

PARTE PRÁCTICA: (15 puntos)

Pregunta 1 (05 puntos) – **Obligatoria (C# – con Visual Studio 2022 – .NET Framework 4.8.1)**

Se ha diseñado una plataforma web en C# con la tecnología ASP .NET y .NET Framework 4.8.1 que permite asignar vehículos registrados en la base de datos a propietarios que también se encuentran registrados. La interfaz gráfica para este propósito se muestra en la Figura 01. Se le ha solicitado completar toda la programación necesaria a nivel de la capa de lógica de negocio (Business) y la capa de presentación (proyecto web) para que esta pantalla pueda funcionar.

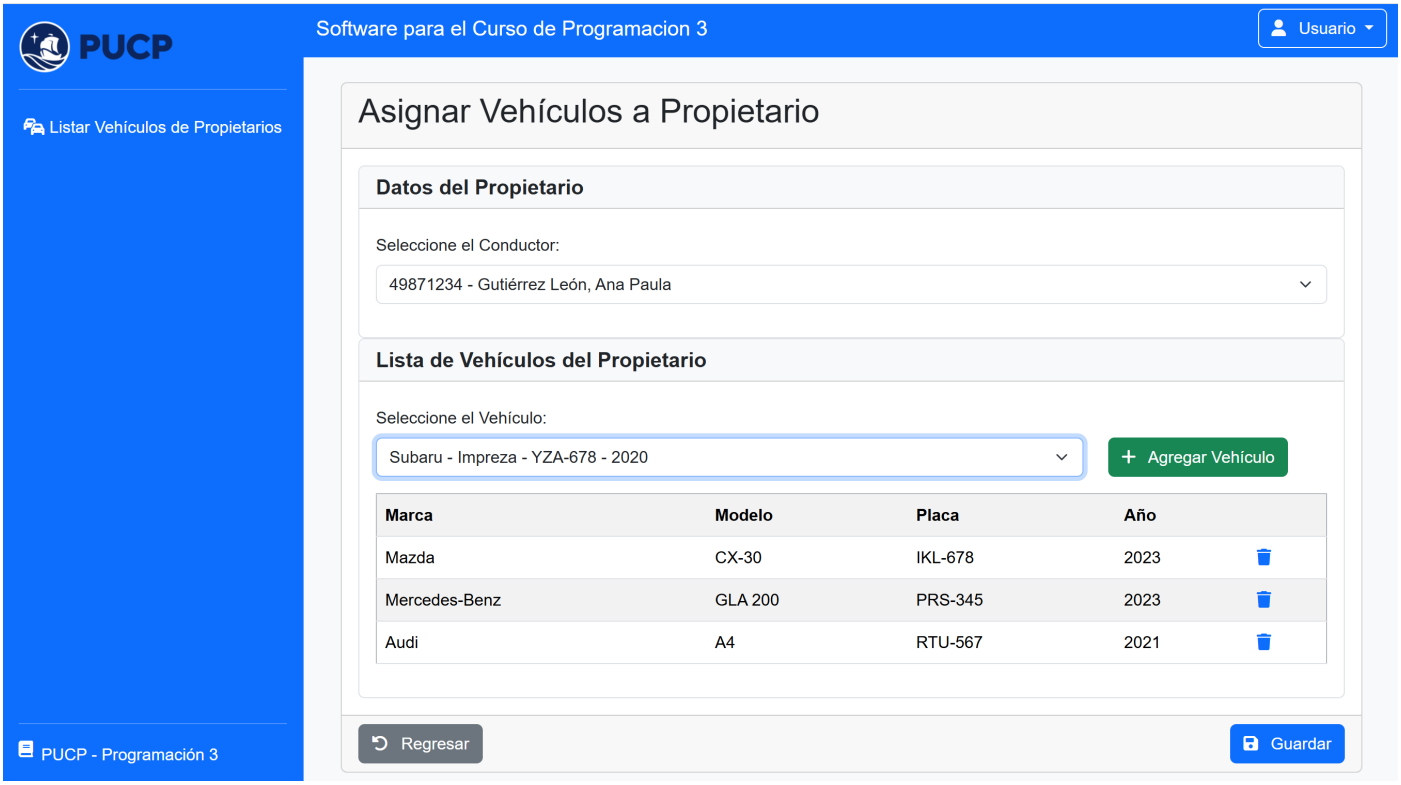

Fig. 01. Formulario para la asignación de vehículos a propietarios

Los métodos de acceso a base de datos ya se encuentran implementados en el ensamblado "DataAccess.dll" que ya se encuentra adjunto al proyecto. Simplemente se deben utilizar los métodos ya programados para lograr el funcionamiento deseado de la interfaz gráfica:

- Al cargar esta pantalla, se mostrará en el primer **combobox**, la lista de propietarios que aún no tienen vehículos asignados. Utilice para este propósito el método `BindingList<Propietario> listarPropietariosSinVehiculoAsignado` de la clase **PropietarioImpl** que recibe como parámetros el *hostname*, el *nombre del esquema*, el *puerto*, el *usuario* y el *password* de acceso a la base de datos. La clase **Propietario** tiene una propiedad llamada "**DNINombreCompleto**" que devuelve los datos del propietario en el formato que se espera en el **combobox**. Asimismo, posee una propiedad "**Id**" que devuelve el valor único del objeto.
- Al cargar esta pantalla, se mostrará en el segundo **combobox**, la lista de vehículos que aún no han sido asignados a propietarios. Utilice para este propósito el método `BindingList<Vehiculo> listarVehiculosSinPropietarioAsignado` de la clase **VehiculoImpl** que recibe como parámetros el *hostname*, el *nombre del esquema*, el *puerto*, el *usuario* y el *password* de acceso a la base de datos. La clase **Vehiculo** tiene una propiedad llamada "**Datos**" que devuelve los datos del vehículo en el formato que se espera en el **combobox**. Asimismo, posee una propiedad "**Id**" que devuelve el valor único del objeto.
- Primero, el usuario seleccionará en el primer **combobox** al propietario al cual desea asignar vehículos. Posteriormente, el usuario irá seleccionando en el segundo **combobox** a los vehículos que desea ir asignando al propietario seleccionado. Una vez identificado el vehículo en el segundo **combobox**, presionará el botón "**Agregar Vehículo**". Este proceso se repite para todos los vehículos que deseamos asignar al propietario y se irá formando una lista en la parte inferior en el componente "**gvVehiculos**", como se muestra en la Figura 01.
- Es posible que, al momento de agregar un vehículo a la lista, el usuario seleccione uno por error. En estos casos, el formulario debe permitir eliminar la fila correspondiente mediante el botón con el ícono de *tacho* asociado a cada registro.
- Una vez que se han agregado todos los vehículos del propietario, el usuario hará clic en el botón "**Guardar**". Se procederá a guardar todas las relaciones del propietario y sus vehículos en la tabla

"**vehiculo_propietario**" de la base de datos. Utilice para esto el método `int registrarAsignacionDeVehiculosAPropietario` que recibe como parámetros el *hostname*, el *nombre del esquema*, el *puerto*, el *usuario*, el *password* de acceso a la base de datos y un objeto Propietario, el cual contiene la lista de vehículos asociados en su atributo "**vehículos**" de tipo **BindingList<Vehiculo>**. Puede utilizar la siguiente estrategia para asociar:

```
propietario.Vehiculos = new BindingList<Vehiculo>(vehiculosSeleccionados.ToArray());
```

No es necesario considerar validaciones.
Coloque sus datos de conexión en el archivo web.config. Utilice el siguiente formato de ejemplo:

```
<add key="db.hostname" value="examen.c6l8fxf9qbzx.us-east-1.rds.amazonaws.com" />
<add key="db.port" value="3306" />
<add key="db.username" value="admin" />
<add key="db.password" value="Urby7vHfNIZW0jBYwTLa" />
<add key="db.database" value="transitsoft" />
```

Además, agregue el código de programación necesario para que la contraseña sea leída de forma encriptada. Sin embargo, deje el valor de la contraseña sin encriptar.
Utilice como base el proyecto en C# que se encuentra en PAIDEIA. Ejecute el script SQL que se adjunta.

Rúbrica:
Lectura de contraseña encriptada: 0.5 punto.
Implementación de listado de propietarios: 1 punto.
Implementación de listado de vehículos: 1 punto.
Implementación de agregar y borrar vehículos al propietario en memoria: 1.5 punto.
Implementación del registro en base de datos: 1 punto.

Pregunta 2 (05 puntos) – **Obligatoria (JAVA – con NetBeans)**

Se requiere procesar y generar infracciones desde las capturas registradas por las cámaras instaladas en tres avenidas de la ciudad. La cámara con código de serie **CAM-LIM-002** se encuentra ubicada en la **Av. Panamericana Sur**, donde la velocidad máxima permitida es de **80 km/h**, y las otras en las avenidas **Javier Prado** y **Universitaria**, cuyas velocidades máximas permitidas son de **50 km/h** respectivamente.
Se le hace entrega de un proyecto Maven con los siguientes módulos:

- **TransitSoftDBManager – Debe implementarse**
- **TransitSoftModelo – Se entrega implementado**
- **TransitSoftPersistencia – Debe implementarse**
- **TransitSoftNegocio – Se entrega semi-implementado**
- **TransitSoftVigilanciaVial – Se entrega implementado**

Con eso se solicita:

1. En la base de datos MySql que se solicitó crear, ejecute los siguientes scripts: **tablas.sql**, **procedimientos.sql**, **inicializacion.sql**
2. Agregar una implementación de DBManager en el módulo TransitSoftDBManager. Este debe ser capaz de conectarse a la base de datos MySql que se solicitó crear.
3. En el módulo TransitSoftPersistencia **definir e implementar** un DAO para Captura. Este DAO **permitirá leer todas las capturas** desde la base datos. Puede usar sentencias SQL o Procedimientos almacenados. **Se recomienda usar el procedimiento listarCapturas disponible en el archivo procedimientos.sql.** Además, este DAO permitirá actualizar el estado de las capturas de **REGISTRADO** a **PROCESADO**.
4. En el módulo **TransitSoftNegocio**, se debe definir la clase de negocio **CapturaBOImpl**, la cual contendrá el método **obtenerCapturasConExcesoDeVelocidad**, este método hará uso de CapturaDAO para obtener todas las capturas. Este método es fundamental, ya que una vez se han obtenido todas las capturas se debe **aplicar polimorfismo** utilizando el **Patrón Estrategia** para determinar si un vehículo ha cometido una infracción por exceso de velocidad, permitiendo una fácil extensión de las reglas sin modificar la clase de negocio. Para seleccionar la lógica de verificación adecuada, el método debe inspeccionar la captura vehicular para obtener el **código de serie de la cámara**, identificando así la avenida de tránsito y el límite de velocidad aplicable. Se solicita definir específicamente las estrategias concretas **EstrategiaViaRegular** y **EstrategiaViaRápida**, que encapsularán las distintas reglas de cálculo para determinar si hubo exceso de velocidad en función de la clasificación de la vía.
5. En la clase CapturaBOImpl implementar el método **actualizar** el cual debe llamar a CapturaDAO de tal manera que pueda actualizar el estado de una captura una vez PROCESADO.

Patrón Estrategia

El Patrón **Estrategia** permite definir una familia de **algoritmos intercambiables**, de modo que el comportamiento de un programa pueda variar fácilmente al seleccionar una **estrategia** (algoritmo) diferente. Su objetivo es cambiar el algoritmo utilizado **sin modificar** el código del programa.

Este patrón se basa en el principio de **separación de responsabilidades** y en el uso de interfaces para lograr un bajo acoplamiento. En lugar de que una clase tenga varios comportamientos mediante estructuras condicionales como if o switch, el comportamiento se **delega a un objeto externo** que implementa una estrategia concreta.

Sus componentes principales son:

1. **Estrategia**: interfaz o clase abstracta que define el método común que usarán todas las estrategias.
2. **Estrategias concretas**: clases que implementan la interfaz y definen versiones específicas del algoritmo.
3. **Contexto**: clase que utiliza una estrategia para ejecutar la operación.

La clase Contexto mantiene una referencia a la interfaz de la estrategia, lo que permite aplicar polimorfismo y lograr el enlace dinámico con el método correspondiente de la estrategia concreta en tiempo de ejecución. Esto significa que el contexto puede funcionar con diferentes estrategias sin necesitar cambios en su código, solo cambiando el objeto que se le pasa como estrategia.

El método definido en la interfaz puede devolver distintos tipos de datos según la operación que se esté realizando; no está limitado a ser void. Esto hace que el patrón sea flexible y aplicable a una amplia variedad de escenarios, desde cálculos y evaluaciones hasta procesos de filtrado o transformación de datos.

```
interface Estrategia {
    boolean evaluar(Object dato);
}

class EstrategiaA implements Estrategia {
    @Override
    public boolean evaluar(Object dato) {
        System.out.println("Evaluando con Estrategia A");
        return true;
    }
}

class EstrategiaB implements Estrategia {
    @Override
    public boolean evaluar(Object dato) {
        System.out.println("Evaluando con Estrategia B");
        return false;
    }
}

class Contexto {
    private final Estrategia estrategia; // Referencia polimórfica

    public Contexto(Estrategia estrategia) {
        this.estrategia = estrategia;
    }

    public boolean ejecutarEstrategia(Object dato) {
        return estrategia.evaluar(dato); // Enlace dinámico al método correcto
    }
}

public class Main {
    public static void main(String[] args) {
        Contexto contexto = new Contexto(new EstrategiaA());
        contexto.ejecutarEstrategia("dato");

        contexto = new Contexto(new EstrategiaB());
        contexto.ejecutarEstrategia("dato");
    }
}
```

Fragmento de Código 1 – Ejemplo de Patrón Estrategia

Pregunta 2 (05 puntos) – **Obligatoria (JAVA o C# – con NetBeans o Visual Studio 2022)**

Dentro del proyecto *TransitSoft*, se cuenta con dos bases de datos relacionales: una en **MySQL** y otra en **Microsoft SQL Server (MSSQL)**. Ambas comparten la **misma estructura de tablas**. Durante el despliegue de la aplicación, algunas sucursales utilizaron la base de datos en MySQL y otras la de MSSQL. Como resultado, la información del sistema se encuentra **distribuida entre ambas bases de datos**, aunque **no existen datos duplicados** entre ellas.

Se le pide que elabore un programa en Java o C# que implemente una regla de negocio para consultar vehículos filtrando por los datos del propietario.

El usuario podrá realizar la búsqueda ingresando los siguientes criterios:

- **DNI**
- **Nombres**
- **Apellidos**

La consulta debe ejecutarse **simultáneamente en ambas bases de datos** y retornar los resultados combinados.

Para resolver esta pregunta, debe cumplir con los siguientes lineamientos:

1. **Los datos de conexión** a ambas bases de datos deben almacenarse en un **archivo externo**.
2. **Las contraseñas** de conexión deben estar **encriptadas** y desencriptarse en tiempo de ejecución.
3. **No se permite duplicar código**. Su solución debe estar **refactorizada** y aplicar principios de reutilización.
4. Debe implementar la solución utilizando el **patrón DAO** enseñado en clase, con una **capa de negocio** que orqueste la lógica de consulta.

En el archivo **Pregunta3_ProyectoBase.zip** se le proporciona una base de proyecto en **Java** que ya implementa la consulta a una sola base de datos. Si lo desea, puede utilizar este proyecto como punto de partida para su solución, extendiéndolo para que consulte ambas fuentes de datos.

Profesores del curso:

Dr. Freddy Paz
Dr. Andrés Melgar
Dr. Eric Huiza

San Miguel, 17 de octubre de 2025