

C#

```
Using System;
Using System.Text.Json;
using namespace SistemaVehicula
```

```
{
```

- 5 Interfaces

```
public interface IVehiculo
```

```
{
```

```
void Encender();
void Apagar();
```

```
}
```

```
public interface IMantenimiento
```

```
{
```

```
void ActivarAlarma();
void DesactivarAlarma();
```

```
}
```

```
public interface INavegacion
```

```
{
```

```
void ActivarGPS();
void DesactivarGPS();
```

```
}
```

```
public interface Icombustible
```

```
{
```

```
void Recargar();
void VerificarNivel();
```

```
}
```

- 5 Clases ABSTRACTAS

```
public abstract class VehiculoBase
```

```
{
```

```
public string NombreVehiculo;
```

```
public VehiculoBase()
```

```
{
```

```
NombreVehiculo = nombre;
```

```
}
```

```
public VehiculoBase(int codigo)
```

```
{
```

```
NombreVehiculo = "Vehiculo numero " + codigo;
```

```
}
```

```
public VehiculoBase (VehiculoBase otro)
{
```

NombreVehiculo = otro.NombreVehiculo;

{

```
public abstract void Describir();
```

{

```
public abstract class MotorBase
```

{

```
public string tipoMotor = "Motor genérico";
```

{

```
public MotorBase (string tipo)
```

{

```
tipoMotor = tipo;
```

{

```
public MotorBase (MotorBase otro)
```

{

```
tipoMotor = otro.tipoMotor;
```

{

```
public abstract void MostrarESpecificaciones();
```

{

```
public abstract class SistemaElectricBase
```

{

```
public string tipoSistema;
```

```
public SistemaElectricBase ()
```

{

```
tipoSistema = "Sistema eléctrico básico";
```

{

```
public SistemaElectricBase (string tipo)
```

{

```
tipoSistema = tipo;
```

{

```
public SistemaElectricobusa (int Voltaje)
{
    tipo Sistema = "Sistema " + Voltaje + " Voltios";
}

public SistemaElectricobase (SistemaElectricobase otro)
{
    tipo Sistema = otro.tipo Sistema;
}

public abstract void Diagnosticar();

public abstract class TransmisionBase
{
    public String tipoTransmision;

    public TransmisionBase ()
    {
        tipo_transmision = "Transmision generica";
    }

    public TransmisionBase (String tipo)
    {
        tipoTransmision = tipo;
    }

    public TransmisionBase (int Velocidades)
    {
        tipoTransmision = "Transmision " + Velocidades + " Velocidades";
    }

    public TransmisionBase (TransmisionBase otro)
    {
    }
```

```
    tipoTransmision = otro.TipoTransmision;
}

Public abstract void CambiarMarcha();
```

```
Public abstract class SistemaFrenosBase
{
    Public String TipoFrenos;
    Public SistemaFrenosBase()
    {
        TipoFrenos = "frenos basicos";
    }

    Public SistemaFrenosBase (String tipo)
    {
        TipoFrenos = tipo;
    }

    Public SistemaFrenosBase (SistemaFrenosBase otro)
    {
        TipoFrenos = otro.TipoFrenos;
    }

    Public abstract void VerificarEstado();
}

Public class CocheInteligente : VehiculoBase, IVehiculo
{
    Public String Marca { get; set; }
    Public String Modelo { get; set; }
    Public String Color { get; set; }
    Public double VelocidadMaxima { get; set; }
    public bool Encendido { get; set; }
```

```

private double VelocidadActual;
private double Combustible;
private UsandoDocumento Configuracion;
private bool GPSActivo;
private byte[] registroAudio;

Protected bool CinturonesAjustados;
protected bool LucesEncendidas;
protected string ModoConduccion;
protected int PuertasCerradas;
protected double Kilometraje;

Public CorrienteInteligente()
:Base("Coche Generico")
{
    Marca = "Generica";
    Modelo = "Base";
    Color = "Azul";
    VelocidadMaxima = 110;
    InicializarDatos();
}

Public CorrienteInteligente(string marca, string modelo, string color, double VelocidadMaxima)
:Base(marca + " " + modelo)
{
    Marca = marca;
    Modelo = modelo;
    Color = color;
    VelocidadMaxima = VelocidadMaxima;
    InicializarDatos();
}

Public CorrienteInteligente(string marca, string modelo, string color, double VelocidadMaxima, double CombustibleInicial)
:Base(marca + " " + modelo)
{
    Marca = marca;
    Modelo = modelo;
    Color = color;
    VelocidadMaxima = VelocidadMaxima;
    InicializarDatos(CombustibleInicial);
}

Public CorrienteInteligente(CorrienteInteligente otro)
:base(otro.NombreVehiculo)
{
}

```

```
Marca = otro. Marca;
Modelo = otro. Modelo;
Color = otro. Color;
VelocidadMaxima = otro. VelocidadMaxima;
VelocidadActual = otro. VelocidadActual;
Combustible = otro. Combustible;
InicializarDatos(otro. Combustible);
```

```
3 Private void InicializarDatos (double valor = 100)
```

```
{ Encendido = false;
VelocidadActual = 0;
Combustible = valor;
gpsActivo = false;
registroAudio = array.Empty<byte>();
Configuracion = XmlDocument.Parse("{\"3\"}");
Macrodireccion = "Normal";
Kilometraje = 0;
CinturonesAbrochados = false;
LucesEncendidas = false;
puertasCerradas = 4;
```

```
3 Public void Encender ()
```

```
{ If (!Encendido)
```

```
{ Encendido = true;
Console.WriteLine("El carro esta encendido");}
```

```
3 Ello
Console.WriteLine("Ya esta encendido");
```

```
3 Public void Apagar ()
```

```
{ Encendido = false;
VelocidadActual = 0;
Console.WriteLine("El carro se apago");}
```

```
3 Public override void Describir ()
```

```
{ Console.WriteLine($"{{NombreVehiculo}} - Marca: {{Marca}}, Modelo: {{Modelo}}, Color: {{Color}}");}
```

Clase 2

Public class MotoCicleta : VehiculoBase, IVehiculob

{

 Public String Marca { get; set; }

 Public String Tipo { get; set; }

 Public Int Cilindraje { get; set; }

 Public double VelocidadMaxima { get; set; }

 Public bool Encendida { get; set; }

 Private double VelocidadActual;

 Private double Combustible;

 Private string[] Accesorios;

 Private bool CaballetActivo;

 Private byte[] historialRaparaciones;

 Protected bool CascoObligatorio;

 Protected bool LucesEncendidas;

 Protected String TipoManeja;

 Protected Int NivelCombustible;

 Protected double Kilometraje;

 Public MotoCicleta ()

 : Base ("MotoCicleta Generico")

{

 Marca = "Generica";

 tipo = "Standard";

 Cilindraje = 125;

 VelocidadMaxima = 115;

 InicializarPatos();

}

 Public MotoCicleta (String marca, String tipo, Int cilindraje, double

 VelocidadMaxima)

 : Base (marca + " " + tipo)

{

 Marca = marca;

 tipo = tipo;

 cilindraje = cilindraje;

 VelocidadMaxima = VelocidadMaxima;

 InicializarPatos();

}

```
public Motocicleta (Motorcycle otra)
:Base (Otro Nombre Vehiculo)
```

```
{  
    Marca = otra.Marca;  
    Tipo = otra.Tipo;  
    Cilindraje = otra.Cilindraje;  
    VelocidadMaxima = otra.VelocidadMaxima;  
    VelocidadActual = otra.VelocidadActual;  
    Combustible = otra.Combustible;  
    InicializarDatos (otra.Combustible);
```

```
3    Private Void InicializarDatos (double Valor = 100)
```

```
{  
    Encendida = false;  
    VelocidadActual = 0;  
    combustible = valor;  
    accesorios = new String [0];  
    CaballeteActivo = true;  
    HistorialReparaciones = array.Emat<bytes> ();  
    CocheObligatorio = true;  
    LucesEncendidas = false;  
    TipoManejo = "Ciudad";  
    NivelCombustible = (int) valor;  
    KIometraje = 0;
```

```
3    public Void Encender ()
```

```
{  
    If (!Encendida && Combustible > 0)  
    {  
        else  
        console.WriteLine ("No se puede encender");
```

```
3    public void Pagar ()
```

```
{  
    Encendido = false;  
    VelocidadActual = 0;  
    Console.WriteLine ("La motocicleta fue apagada");
```

```
3    public override void Describir ()
```

```
{  
    Console.WriteLine ("{NombreVehiculo} - Marca: {Marca} Cilindraje:  
    {cilindraje} cc, VelocidadMaxima: {VelocidadMaxima} Km/h");
```

Camion

Public class Camion : Vehiculo Basa, IMantenimiento

{

Public String Marca {get; set;}

Public String tipoCarga {get; set;}

Public double CapacidadCarga {get; set;}

Public int NumeroEjes {get; set;}

Public bool Enfuncionamiento {get; set;}

Private double pasoActual;

Private double Combustible;

Private string reparacionesPendientes;

Private bool necesitaMantenimiento;

Private bool documentoCarga;

Protected bool RemolqueConectado;

Protected bool SistemaHidraulico;

Protected String tipoMotor;

Protected int HorasUsadas;

Protected double Kilometraje;

Public Camion ()

:Base ("Camion Generico")

{

Marca = "Generico";

TipoCarga = "Generico";

CapacidadCarga = 5000;

NumeroEjes = 2;

InicializarDatos();

}

Public Camion (String marca, String tipoCarga, double Capacidad, int ejes)

:base (marca + " Camion")

Marca = Marca;

TipoCarga = TipoCarga;

CapacidadCarga = Capacidad;

NumeroEjes = ejes;

InicializarDatos();

}

```
public Camion ( string marca, string tipoCarga, double capacidad, int ejes, double combustibleInicial )
```

```
: base ( marca + " Camion" )
```

```
{
```

```
Marca = marca;
```

```
TipoCarga = tipoCarga;
```

```
CapacidadCarga = capacidad;
```

```
NumerosEjes = ejes;
```

```
InicializarDatos ( combustibleInicial );
```

```
}
```

```
public Camion ( Camion otro )
```

```
: base ( otro.NombreVehiculo )
```

```
{
```

```
Marca = otro.Marca;
```

```
TipoCarga = otro.TipoCarga;
```

```
CapacidadCarga = otro.CapacidadCarga;
```

```
NumerosEjes = otro.NumerosEjes;
```

```
PesoActual = otro.PesoActual;
```

```
Combustible = otro.Combustible;
```

```
InicializarDatos ( otro.Combustible );
```

```
}
```

```
private void InicializarDatos ( double valor = 200 )
```

```
{
```

```
Enfuncionamiento = false;
```

```
PesoActual = 0;
```

```
Combustible = valor;
```

```
reparacionesPendientes = new String [ 0 ];
```

```
necesitoMantenimiento = false;
```

```
documentoCarga = array.Empleado < byte > [ 0 ];
```

```
RevolucionMotor = false;
```

```
SistemaHidraulico = true;
```

```
TipoMotor = " Diesel ";
```

```
HorasUso = 0;
```

```
Kilometraje = 0;
```

```
}
```

```
public void RealizarMantenimiento ()
```

```
{
```

```
necesitoMantenimiento = false;
```

```
Console.WriteLine ( " Mantenimiento realizado al camion " );
```

```
}
```

```

public void CambiaAceite()
{
    console.WriteLine (" Aceite del Camion Cambiado ");
}

public override void Pascibir ()
{
    console.WriteLine ($ { NombreVehiculo } - Marca: ${ Marca }, Capacidad:
    ${ capacidadCarga } Kg, Ejes : ${ NumeroEjas } " );
}

public class AutoDeportivo : VehiculoBase , ISeguridad
{
    public string Marca { get ; set ; }
    public string Modelo { get ; set ; }
    public int potencia { get ; set ; }
    public double Aceleracion { get ; set ; }
    public bool Encendido { get ; set ; }

    private double VelocidadActual ;
    private double nivelTurbo ;
    private string[] modosDePista ;
    private bool AlarmaActiva ;
    private byte[] Telemetria ;
    protected bool ModoDeportivo ;
    protected bool AlarmaDesplegada ;
    protected string SuspensionActiva ;
    protected int RPMActual ;
    protected double TemperaturaMotor ;

    public AutoDeportivo ()
        : Base (" Auto Deportivo Generico ")

    {
        Marca = "Generico" ;
        Modelo = "Sport" ;
        Potencia = 300 ;
    }
}

```

```

    Aceleracion = 5.5;
    InicializarDatos();
}

Public AutoDeportivo (string marca, modelo, Impotencia, double aceleracion,
double turbos) {
    Base (marca + " " + modelo)

    {
        Marca = marca;
        Modelo = modelo;
        potencia = potencia;
        Aceleracion = aceleracion;
        InicializarDatos (turbos);
    }

    public AutoDeportivo (AutoDeportivo otro)
    : Base (otro.NombreVehiculo)

    {
        Marca = otro.Marca;
        Modelo = otro.Modelo;
        potencia = otro.potencia;
        Aceleracion = otro.Aceleracion;
        velocidadActual = otro.velocidadActual;
        nivelTurbo = otro.nivelTurbo;
        InicializarDatos (otro.nivelTurbo);
    }

    private void InicializarDatos (double turbo = 0)
    {
        Encendido = false;
        velocidadActual = 0;
        nivelTurbo = turbo;
        modosDisponibles = new string [] { "Circuito", "Drag", "Drift" };
        alarmaActiva = false;
        telemetria = Array.Empty<byte> ();
        ModoDeportivo = false;
        AleronDespegado = false;
        SuspensionActiva = "Confort";
        RPMActual = 0;
        TemperaturaMotor = 20;
    }

    public void ActivarAlarma ()
    {
        alarmaActiva = true;
        Console.WriteLine ("Alarma del Auto activada");
    }
}

```

```
public void DesactivarAlarma ()  
{  
    alarmaActivo = false;  
    console.WriteLine ("Alarma desactivada");  
}  
  
public override void Describir ()  
{  
    console.WriteLine ($" {NombreVehiculo} - {Marca} {Marca}, {potencia}  
    {potencia} HP, 0-100 : {Aceleracion} Seg ");  
}  
  
public class AutoElectrico : VehiculoBase, ICombustible  
{  
    public string Marca { get; set; }  
    public string Modelo { get; set; }  
    public int Autonomia { get; set; }  
    public double TiempoCarga { get; set; }  
    public bool Encendido { get; set; }  
    public double bateriaActual { get; set; }  
    private double EficienciaEnergética;  
    private string estacionesCarga;  
    private bool CargadoRápido;  
    private byte historialCargas;  
    protected bool ModoEco;  
    protected bool RegeneraciónFrenada;  
    protected string TipoBatería;  
    protected int CiclosRecarga;  
    protected double ConsumoPromedio;  
    public AutoElectrico ()  
        : Base ("Auto Electrónico Generico")  
    {
```

```
Marca = "behaviorica");
Modelo = "N";
Autonomia = 300;
TipoCarga = 8;
InicializarDatos();
```

```
3 public AutoElectrico (String marca, String modelo, int autonomia,
double tiempoCarga
: Base (marca + " " + modelo)
```

```
Marca = marca;
Modelo = Modelo;
Autonomia = autonomia;
TiempoCarga = tiempoCarga;
InicializarDatos = (bateriaActual);
```

```
3 public AutoElectrico (AutoElectrico otro)
```

```
: Base (otro. NombreVehiculo
```

```
{
```

```
Marca = otro.Marca;
Modelo = otro.Modelo;
Autonomia = otro.Autonomia;
TiempoCarga = otro.TiempoCarga;
bateriaActual = otro.bateriaActual;
eficienciaEnergетica = otro.eficienciaEnergетica;
InicializarDatos (otro.bateriaActual);
```

```
8
```

```
private void InicializarDatos (double bateria = 100)
```

```
{
```

```
Encendido = false;
bateriaActual = bateria;
eficienciaEnergетica = 95;
estacionesCargadas = arrayEmpty <byte> (1);
ModoEco = true;
RegeneracionFrenado = true;
tipoBateria = "Lithium-ion",
ciclosRecarga = 0,
consumoPromedio = 15;
```

```
3 public void Recargar ()
```

```
{ bateriaActual = 100;
ciclosRecarga++;
console.WriteLine ("Auto electrico recargado completamente");
```

```

    Public void VolverAlNivel()
    {
        Console.WriteLine($" {NombreVehiculo} - Marca {Marca}, Autonomia: {Autonomia} Km, Tiempo Carga: {TiempoCarga} h");
    }
}

FARMACIA C#
Using System;
Using System.Text.Json;
namespace SistemaFarmaceutico
{
    public interface INegocio
    {
        void abrir();
        void Cerrar();
    }

    public interface IVentas
    {
        void RegistrarVenta();
        void cancelarVenta();
    }

    public interface IInventario
    {
        void AgregarProducto();
        void QuitarProducto();
    }

    public interface IEmpleados
    {
        void contratar();
        void despagar();
    }

    public interface IProveedor
    {
        void RealizarPedido();
        void RecibirPedido();
    }
}

```

ABSTRACTA

```
public abstract class EstablecimientoBase
{
    public String NombreEstablecimiento;
    public EstablecimientoBase()
    :this("EstablecimientoBaseGenerico", 0) {}

    public EstablecimientoBase(String nombre)
    :This(nombre, 0) {}

    public ("EstablecimientoBase (String nombre)
    :This (nombre, 0) {}

    public EstablecimientoBase (String nombre, int codigo)
    {
        nombreEstablecimiento = (codigo > 0) ? "Establecimiento numero"
        {codigo} - {nombre} : nombre;
    }

    public EstablecimientoBase (Establecimiento otro)
    :This (otro.NombreEstablecimiento, 0) {}

    public abstract void MostrarInformacion();
}

public abstract class ProductoBase
{
    public String NombreProducto;
    public ProductoBase()
    :this("Producto Generico", 0) {}

    public ProductoBase (String nombre)
    :This(nombre, 0) {}

    public ProductoBase (String nombre, int codigo)
    {
        NombreProducto = (codigo > 0) ? "producto codigo" {codigo} -
        {nombre} : nombre;
    }
}
```

```
public productoBase (productoBase otro)
: This (otro.NombreProducto, 0) {}  
public abstract void MostrarDetalles ();  
}  
public abstract class personalBase  
{  
    public String NombrePersonal;  
    public personalBase ()  
    : This ("Personal sin nombre", 0) {}  
    public personalBase (String nombre)  
    : This (Hombre, 0) {}  
    public personalBase (String nombre, int legajo)  
    {  
        nombrePersonal = (legajo > 0) ? "Personal legajo " + legajo : "  
        { nombre }";  
    }  
    public personalBase (personalBase otro)  
    : This (otro.NombrePersonal, 0) {}  
    public abstract void MostrarDatos ();  
}  
public String TipoServicio;  
public ServicioBase ()
: This ("Servicio basico", 0) {}  
public ServicioBase (String tipo)  
: This (tipo, 0) {}  
public ServicioBase (String tipo, int nivel)  
{  
    TipoServicio = (nivel > 0) ? "Servicio nivel " + nivel : " " + tipo;  
}
```

```

public abstract void Ejecutar();
}

public abstract class SistemaBase
{
    public String NombreSistema;
    public SistemaBase()
    {
        this("Sistema Generico", 0);
    }
    public SistemaBase(String nombre, int Version)
    {
        NombreSistema = (Version > 0) ? "Sistema Version " + Version : nombre;
    }
    public SistemaBase(SistemaBase otro)
    {
        this(otro.NombreSistema, 0);
    }
    public abstract void Inicializar();
}

Clase Normal
public class farmacia : EstablecimientoBase(Negocio)
{
    public String nombre { get; set; }
    public String Direccion { get; set; }
    public String Encargado { get; set; }
    public bool Abierta { get; set; }
    public int NumeroEmpleados { get; set; }
    private double IngresosDiarios;
    private int VentasRealizadas;
    private bool SistemaActivo;
    private byte LogoFarmacia;
    private XmlDocument RegistroVentas;
    protected double precioPromedio;
    protected int productoDisponibles;
    protected bool TurnoNoche;
    protected string TipoFarmacia;
    protected string proveedorPrincipal;
}

```

```
public farmacia()
: This ("farmaciabenito", "Sin Direccion", "Sin encargo", 3) {}
public farmacia (String nombre, String direccion, String encargado)
: This (nombre, direccion, encargado, 3) {}

public farmacia (String nombre, String direccion, String encargado, int empleados)
```

: Base (nombre)

{

```
Nombre = nombre;
Direccion = direccion;
Encargado = encargado;
NumeroEmpleados = Empleados;
```

inicializarDatos();

}

```
public farmacia (farmacia otro)
```

: This (otro.Nombre, otro.Direccion, otro.Encargado, otro.NumeroEmpleados)

{

```
IngresosDiaridos = otro.IngresosDiaridos;
VentasRealizadas = otro.VentasRealizadas;
SistemaActivo = otro.SistemaActivo;
precioPromedio = otro.PrecioPromedio;
```

}

```
private void inicializaDatos ()
```

{

```
Abierta = false;
IngresosDiaridos = 0;
VentasRealizadas = 0;
SistemaActivo = true;
LocoFarmacia = new byte[0];
PrecioPromedio = 250000;
ProductosDisponibles = 1000;
turnoNoche = false;
TipoFarmacia = "popular";
ProveedorPrincipal = "distribuidora";
```

}

```
public void Abrir ()
```

{

```
Abierta = true;
Console.WriteLine ("Abierta");
```

}

```

public void cerrar()
{
    Abierto = false;
    Console.WriteLine("Cerrado");
}

Implementa de clase abstracta

public override void MostrarInformacion()
{
    Console.WriteLine($"farmacia : {NombreEstablecimiento} - encargado : "
    $"{encargado}");
}

public class Drogueria : EstablecimientoBase, INegocio
{
    public string Ciudad { get; set; }
    public string Gerente { get; set; }
    int Sucursales { get; set; }
    private double CapitalInvertido;
    private int ClientesMensuales;
    protected double MargenBancario;

    public Drogueria()
    {
        this(., ., "Sin Ciudad", "Sin Gerente", 1);
    }

    public Drogueria(string nombre, string ciudad, string gerente)
    {
        this(nombre, ciudad, gerente, 1);
    }

    public Drogueria(string nombre, string ciudad, string gerente,
    int sucursales)
    {
        Ciudad = ciudad;
        Gerente = gerente;
        Sucursales = sucursales;
        inicializarDatos();
    }

    public Drogueria(Drogueria otra)
    {
        otro.NombreEstablecimiento = otra.NombreEstablecimiento;
        otra.Ciudad = otra.Ciudad;
        otra.Gerente = otra.Gerente;
        otra.Sucursales = otra.Sucursales;
    }
}

```

```

    } CapitalInvertido = otra.CapitalInvertido;
    } private void inicializarDatos () {
    {
        CapitalInvertido = 0;
        ClientesMensuales = 0;
        MargenGanancia = 30;
    }
    } public void Abrir () =>
        Console.WriteLine ("Operativo");
    public void Cerrar () => Console.WriteLine ("Cerrado");
    public override MostrarInformacion()
    {
        Console.WriteLine ($"Nombre Establecimiento: {ciudad}, {ciudad}, {ciudad}");
        Sucursales {Sucursales} ";
    }
    } public class Medicamento : ProductoBase, IInventario
    {
        public string NombreComercial { get; set; }
        public string Laboratorio { get; set; }
        public double Precio { get; set; }
        private int UnidadesStock { get; set; }
        private string Lote;
        protected int DiasVencimiento;
        public Medicamento ()
        :This ("Generico", "Sin Clas", 00, 100) {}
        public Medicamento (string nombre, string laboratorio,
double precio)
        :This (nombre, laboratorio, precio, 100) {}
        public Medicamento (string nombre, string laboratorio,
double precio, int stock)
        :Base (nombre)
    }

```

```
NombreComercial = nombre;
laboratorio = laboratorio;
precio = precio;
InicializarDatos (stock);
```

}

```
public Medicamento (Medicamento otro)
```

```
:this (otro.NombreComercial, otro.laboratorio, otro.precio,
otro.UnidadesStock)
```

{

```
Lote = otro.Lote;
```

}

```
private void InicializarDatos (int stock)
```

{

```
UnidadesStock = stock;
```

```
Lote = "LOTE - DEF";
```

```
DiasVencimiento = 365;
```

}

```
public void AgregarProducto ()
```

{

```
UnidadesStock += 50;
```

```
Console.WriteLine ("Incrementado");
```

}

```
public void QuitarProducto ()
```

{

```
if (UnidadesStock > 0)
```

{

```
UnidadesStock --;
```

```
Console.WriteLine ("{" + NombreProducto + " - precio: " + precio + " unidades: " + stock + " Lote: " + Lote + " Vencimiento: " + DiasVencimiento + " dias");
```

}

}

```
public override void MostrarDetalles ()
```

{

```
Console.WriteLine ("{" + NombreProducto + " - precio: " + precio + " unidades: " + stock + " Lote: " + Lote + " Vencimiento: " + DiasVencimiento + " dias");
```

}

}

```
public class farmaceutico : personalBasico, empleado
```

```
{
```

```
    public string Nombre { get; set; }
```

```
    public string Especialidad { get; set; }
```

```
    public double Salario { get; set; }
```

```
    private string TarjetaProfesional;
```

```
    private int horasTrabajadas;
```

```
    protected string Turno;
```

```
    public farmaceutico()
```

```
: This ("Generico", "General", 2000000, "TP-000") {}
```

```
    public farmaceutico()
```

```
: This (string nombre, string especialidad, double salario, string tarjeta)
```

```
: Base (nombre)
```

```
{
```

```
    Nombre = nombre;
```

```
    Especialidad = especialidad;
```

```
    Salario = salario;
```

```
    InicializarDatos (tarjeta);
```

```
}
```

```
    public farmaceutico (farmaceutico otro)
```

```
: This (otro.Nombre, otro.Especialidad, otro.Salario, otro.
```

```
TarjetaProfesional)
```

```
{
```

```
    horasTrabajadas = otro.horasTrabajadas;
```

```
}
```

```
    private void InicializarDatos (string tarjeta)
```

```
{
```

```
    TarjetaProfesional = tarjeta;
```

```
    horasTrabajadas = 0;
```

```
    Turno = "Diurno";
```

```
}
```

```
    public void Contratar () => Console.WriteLine ("Contratado");
```

```
    public void Despedir () => Console.WriteLine ("Despedido");
```

```

public override void MostrarDatos()
{
    Console.WriteLine($"{{ NombrePersonal }} - Especialidad: {{ Especialidad }},\nSalario: {{ Salario }}");
}

Public class proveedor : EstablecimientoBase, IProveedor
{
    public string NombreEmpresa { get; set; }
    public string Contacto { get; set; }
    public int TiempoEntrega { get; set; }
    private double MontoFacurado;
    private int pedidosRealizados;
    protected string Calificacion;

    public proveedor()
    :This("Generica", "sin Contacto", 7) {}

    public proveedor(string Empresa, string Contacto)
    : this(Empresa, Contacto, 7) {}

    public proveedor(string Empresa, string Contacto, int TiempoEntrega)
    : this(Empresa, Contacto, TiempoEntrega, 0)

    Base(Empresa)
    {
        NombreEmpresa = Empresa;
        Contacto = Contacto;
        TiempoEntrega = TiempoEntrega;
        InicializarDatos();
    }

    public proveedor Iproveedor Otro
    : This(Otro.NombreEmpresa, Otro.Contacto, Otro.TiempoEntrega)
    {
        montoFacurado = Otro.montoFacurado;
    }

    private void InicializarDatos()
    {
    }
}

```

Monto facturado = 0;
pedidos Realizados = 0;
Calificación = "A +";

3

```
public void RealizadosPedidos()  
{  
    pedidos Realizados ++;  
    Console.WriteLine("realizada al proveedor.");  
}  
  
public void RecibirPedido()  
{  
    Console.WriteLine("pedido recibido del proveedor.");  
}
```

3
public override void MostrarInformacion()

```
{  
    Console.WriteLine($"{{ nombre }} - {{ contacto }}\nentrega: {{ tiempo Enrega }} dias");  
}
```

3
Sistema hospitalario

```
public interface INadmission { void Admitir(); }
```

```
public interface ITratamiento { void Tratar(); }
```

```
public interface IEmergencia { void Trigger(); }
```

```
public interface IRecursos { void Solicitar(); }
```

```
public interface IHorarios { void AsignarTurno(); }
```

```
public string NombreCompleto;
```

```
public personaHospitalBase()  
: This("apep", 0) {}
```

```
public personaHospitalBase(string nombre)  
: This(nombre, 0) {}
```

```
public personaHospitalBase(string nombre, int id)
```

```
{
```

```
    NombreCompleto = (id > 0) ? $"{ID} {nombre}" : nombre;
```

3

```

public Person HospitalBase (Person HospitalBase otro) {
    if (otro.NombreCompleto == null)
        return this;
    else
        return new Person(otro.NombreCompleto, otro);
}

public abstract void ModificarInfo();
}

public abstract class DepartamentoBase {
    public String NombreDepartamento;
    public DepartamentoBase () {
        this("Generico", "GNR");
    }
    public DepartamentoBase (String nombre) {
        this(nombre, "GNR");
    }
    public Departamento (String nombre, String codigo) {
        NombreDepartamento = (codigo != "GNE") ? codigo : nombre;
        nombre = NombreDepartamento;
    }
    public DepartamentoBase (DepartamentoBase otro) {
        this(otro.NombreDepartamento, "GNR");
    }
    public abstract void MostarProtocolos();
}

public abstract class EquipoMedicoBase {
    public String NombreEquipo;
    public EquipoMedicoBase () {
        this("Equipo Generico");
    }
    public EquipoMedicoBase (String nombre) {
        this(nombre, 0);
    }
    public EquipoMedicoBase (String nombre, int noSerie) {
        NombreEquipo = (noSerie > 0) ? $"Serie {noSerie}" : nombre;
    }
}

```

```

public EquipoMedicoBase (EquipoMedicoBase otro) {
    this (otro.NombreEquipo, 0) {}
}

public abstract void VerificarEstado();
```

}

```

public String NombreComercial;
```

```

public MedicamentoBase ()
```

```

    this ("Farmaco Generico", 0) {}
```

```

public MedicamentoBase (String nombre, int dosisMg)
```

{

```

    NombreComercial = (dosisMg >= 0) ? "{'nombre'} ({dosisMg} mg)" : nombre;
```

}

```

public MedicamentoBase (MedicamentoBase otro)
```

```

    this (otro.NombreComercial, 0) {}
```

```

public abstract void MostrarAdvertencias();
```

{

```

public abstract class CamuBase
```

{

```

    public String NumeroCamu;
```

```

    public CamuBase ()
```

```

        this ("No asignado", 0) {}
```

```

    public CamuBase (String numero)
```

```

        this (numero, 0) {}
```

```

    public CamuBase (String numero, int peso)
```

{

```

        NumeroCamu = (peso > 0) ? (camu.{'numero'} - peso) : numero;
```

}

```

public CamuBase (camuBase otra)
```

```

    this (otra.NumeroCamu, 0) {}
```

```

public abstract void ModificarCupacion();
```

}

```

public class paciente : PersonaHospitalizada, Admision
{
    public string Sintoma { get; set; }
    private bool Hospitalizado;
    public paciente()
    {
        This["NN", "Generico", 30];
    }
    public paciente (String nombre)
    {
        This[nombre, "Polar", 30];
    }
    public paciente (string nombre, string sintoma, int edad)
    {
        Base(nombre, edad);
        Sintoma = sintoma;
        inicializarDatos();
    }
    public paciente (paciente otro)
    {
        This[otro.NombreCompleto, otro.Sintoma, 0];
    }
    private void inicializarDatos()
    {
        Hospitalizado = false;
    }
    public void Admitir ()
    {
        Hospitalizado = true;
        Console.WriteLine($"[{ADMISSION}] {NombreCompleto} admitido por {Sintoma}");
    }
    public override void MostrarInfo()
    {
        Console.WriteLine($"[{paciente}] {NombreCompleto} {Hospitalizado ? "Hospitalizado" : "No es Alta"}");
    }
}
public class Medico : PersonaHospitalizada, Tratamiento
{
}

```

```

public string Especialidad { get; set; }
private bool deTurno;
public Medico()
{
    This ("Dr. Generic", "General", 0) {};
    public Medico (string nombre)
    {
        This (nombre, "General", 0) {};
    }
    public Medico (string nombre, string especialidad, int pacientes)
    {
        Base (nombre, pacientes);
        Especialidad = especialidad;
        InicializarDatos ();
    }
}
public Medico (Medico otro)
{
    This (otro.Nombre, otro.Completo, otro.Otro, especialidad, 0) {};
    private void InicializarDatos ()
    {
        deTurno = true;
    }
    public void Tratar ()
    {
        Console.WriteLine ($"El Médico: ${NombreCompleto}, de Turno: ${deTurno}");
    }
}
public class SalaEmergencia (DepartamentoBase, Emergencia)
{
    public int ComillasOcupadas { get; set; }
    private int MedicosPresentes;
    public SalaEmergencia ()
    {
        This ("Urgencias", 0) {};
    }
    public SalaEmergencia (string nombre)
    {
        This (nombre, 0) {};
    }
    public SalaEmergencia (string nombre, int Capacidad)
    {
        Base (nombre, "EM");
    }
}

```

```

Camillas Ocupadas > Capacidad;
Iniciador de computadora ( )
}

public SalaEmergencia (SalaEmergencia otra)
{
    This (otra.NombreDepartamento) { } ;
    private void InicializarDtos () { } ;
    medicosPresentes = 3, } ;

public void friage()
{
    CamillasOcupadas += 1;
    Console.WriteLine ($" [EMERGENCIA] Paciente en friage");
    Ocupacion = { CamillasOcupadas } ;
}

public override void MostrarProtocolos ()
{
    Console.WriteLine ($" Departamento: {NombreDepartamento}");
    MedicosPresentes } ;
}

public class MonitorCardiaco : EquipoMedicoBase, IRecorrido
{
    public string Modelo { get; set; }

    private bool encendido;

    public MonitorCardiaco ()
    {
        This ("Monitor", "XYZ", 0) { } ;
    }

    public MonitorCardiaco (String nombre)
    {
        This (nombre, "XYZ", 0) { } ;
    }

    public MonitorCardiaco (String nombre, String modelo, int bateria)
    {
        This (nombre, 0);

        Modelo = Modelo;
        InicializarDtos ();
    }

    public MonitorCardiaco (MonitorCardiaco otro)
    {
        This (otro.NombreEquipo, otro.Modelo, 0) { } ;
    }

    private void InicializarDtos ()
    {
        encendido = true;
    }
}

```

```

public void solicitar()
{
    Console.WriteLine($"[RECURSOS] {NombreEquipo}
Solicita Revision Técnica."); }

public override void VerificarEstado()
{
    Console.WriteLine($"[{Equipo}:{NombreEquipo}], Modelo:
{Modelo}, Encendido: {Encendido}"); }

public class Enfermera : PersonaHospitalBase, IHorarios
{
    public string AreaAsignadas { get; set; }
    private string turno;

    public Enfermera()
        : base("Enfermera", "UGI", 0) {}

    public Enfermera(string nombre)
        : base(nombre, "UCI", 0) {}

    public Enfermera(string nombre, string area, int experiencia)
        : base(nombre, experiencia) {}

    public void AsignarTurno()
    {
        areaAsignada = area;
        InicializarPuestos();
    }

    public Enfermera(Enfermera otra)
        : base(otra.Nombre, otra.AreaAsignada, 0) {}

    private void InicializarPuestos()
    {
        turno = "Turno";
    }

    public void AsignarTurno()
    {
    }
}

```

turno = "Nocturno";
Console.WriteLine (d, "El turno es") { Nombre }
Sembra a turno { turno } . "

}

public override void MostrarUE()

{

Console.WriteLine (d, "Nombre del turno:
{ turno } , Area { AreavAreyabu } , ");

}

B

GO

```
package main
```

```
import (
```

```
    "encoding/json"
```

```
    "fmt"
```

```
    "time"
```

```
Type EncendidoApagado Interface {
```

```
    Encender() Celular
```

```
    ApagarCelular()
```

```
}
```

```
Type GestionEnergia Interface {
```

```
}
```

```
Type GestionEnergia Interface {
```

```
    CargaBateria()
```

```
}
```

```
Type Configurable Interface {
```

```
    SetConfiguracion(Config map[string]Interface{})
```

```
    GetConfiguracion() map[string]Interface{}  
}
```

```
}
```

```
Type DispositivoBase struct {
```

```
    Marca string
```

```
    Modelo string
```

```
    SistemaOperativo string
```

```
    Almacenamiento int
```

```
    VersionSoftware string
```

```
}
```

```
func NewDispositivoBase (marca, modelo, so string,  
almacenamiento int) DispositivoBase {
```

```
    return DispositivoBase{
```

```
        Marca: Marca,
```

```
        Modelo: Modelo,
```

```
        SistemaOperativo: SO,
```

```
        Almacenamiento: Almacenamiento,
```

```
        VersionSoftware: "4.0.0",
```

```
}
```

```
Type ControlCelular struct {
```

```
    dispositivoDispositivoBase
```

```
    Encendido bool
```

tiempoButon	int
ConectadoWifi	local
CódigoDestruyed	String
registroBindInt	[] byte
Configuracion	map [String] Interface { }
fechaUltimaCarga	time, time
modoOscuro	bool
temperatura	float64
listaAplicaciones	[] string

func (* ControlCelular) IniciarizarEstadoPrivado () {

```

C. Encendido = false
C. nuevoButon = 00
C. ConectadoWifi = false
C. CódigoDestruyed = "0000"
C. ModoOscuro = false
C. temperatura = 28.5
C. listaAplicaciones = [ ] string { "Telefono", "Mensajes",
"Camara" }
C. registroBindInt = make [ ] byte, 0 )

```

fun NuevoControlCelularFull (marca, modelo, so, string,
almacenamiento int) * ControlCelular {
 dispositivosBase := NewDispositivosBase (marca, modelo, so,
almacenamiento),
}

Cell. IniciarizarEstadoPrivado ()
return Cel

func NuevoControlCelularGenerico () * ControlCelular { }

return NuevoControlCelularFull (marca, modelo, so, almacenamiento)

func NuevoControlCelularGenerico () * ControlCelular {
 return NuevoControlCelularFull ("Generico", "Modelo Basico",
"Basicos", 64)

func NuevoControlCelularCopy (original * ControlCelular) *
ControlCelular { }

Copy := NuevoControlCelularFull (

original . SistemaOperativo

original . Marca

original . Modelo

original . Almacenamiento,

}

```

Copia. Encendido = Original. Encendido
Copia. nivelBateria = Original. nivelBateria
Copia. ConectadoWiFi = Original. ConectadoWiFi
Copia. CódigoBluetooth = Original. CódigoBluetooth
Copia. fechaUltimaCarga = Original. fechaUltimaCarga
Copia. temperatura = Original. temperatura
Copia. registroBinario = Objeto (byte[])
Copia. listaAplicaciones = Objeto (String), original. listaAplicaciones
IF Original. configuracion != null {
    Copia. configuracion = new Map<String, Interface>
    for K, V := range Original. configuracion {
        Copia. configuracion[K] = V
    }
}
return Copia
}

func (C *ControlCelular) EncenderCelular () {
    IF C. Encendido = True
        fmt.Println ("Ya -> Encendido Celular.", "\n", "(Marca, C. Marca)
    }
    func (C *ControlCelular) ApagarCelular () {
        IF ! C. Encendido {
            fmt.Println ("Apagado")
        }
    }
    C. Encendido = false
    fmt.Println ("Se ha apagado")
}

func (C *ControlCelular) CargaBateria () {
    C. nivelBateria = 100
    C. fechaUltimaCarga = time.Now ()
    fmt.Println ("Carga al +00% (1.5)\n")
    C. fechaUltimaCarga.Format ("%")
}

```

```
func (C* ControlCelular) ConectarWifi (red string) {  
    C. ConectadoWifi = true  
    fmt.Println ("Conectado a la red wifi \"", s, "\n", red)
```

```
func (C* ControlCelular) AbrirAplicacion (nombre string) {  
    if !C. Encendido {  
        fmt.Println ("No se puede abrir")  
        return
```

```
    for -app : - range C. listaAplicaciones {  
        if app == nombre {  
            fmt.Printf ("Abriendo app %d\n", nombre)  
            C. nivelBateria -= 2  
            return
```

```
    fmt.Printf ("La aplicacion \"%s\" no instalada\n", nombre)
```

```
func (C* ControlCelular) MostrarEstado () {  
    fmt.Println ("Estado")  
    fmt.Println ("Marca: ", C. Marca, "Modelo: ", C. Modelo, "SO: ", C. SistemaOperativo, "Versión: ", C. VersiónSoftware)  
    fmt.Printf ("Encendido: %t | Batería: %d/%d | WiFi: %t\n",  
        C. Encendido, C. nivelBateria, C. ConectadoWifi)  
    fmt.Printf ("Modo Oscuro: %t | Temperatura: %f °C\n",  
        C. ModoOscuro, C. Temperatura)
```

```
func (C* ControlCelular) MostrarConfiguracion () {
```

```
    if C. Configuracion == nil {
```

```
        fmt.Println ("No configuración disponible")  
        return
```

```

data, j := JSON.MarshalIndent(c.Configuration, "", " ")
fmt.Println("Configuracion actual:")
fmt.Println(string(data))
}

func (c *ControlCelular) SetConfiguration(config map[string]any)
interface {
    c.Configuration = config
}

func (c *ControlCelular) SetConfiguration(config map[string]any)
interface {
    c.Configuration = config
}

func (c *ControlCelular) GetConfiguration() map[string]any
interface {
    return c.Configuration
}

}

// Habitad

type Ejecutable interface {
    ejecutar() string
    verificarEstado() string
}

type EntidadBase struct {
    ID      string
    fechaCreacion time.Time
}

func (e *EntidadBase) getInfoBase() string {
    return
}

fmt.Sprintf("ID: %s, creado: %s", e.ID, e.fechaCreacion)
format("2006-01-02")

}

type ReporteAuditoria struct {
    EntidadBase
    NombreAuditor string
    Sistema       string
    puntuacionSeguridad float64
    completo      bool
}

```

```
func (r *ReporteAuditoria) Ejecutar () string {
    if r.completado {
        return
    }
    fmt.Sprintf (" Reporte %s ", r.sistema)
}
```

```
func (r *ReporteAuditoria) VerificarEstado () string {
    estatus := "pendiente"
    if r.completado {
        estatus = "finalizado"
    }
}
```

```
return
fmt.Sprintf ("%s actual: %.2f . Puntuacion: %.1f", estatus,
    r.puntuacionSeguridad)
```

```
}
```

```
func NewReporteAuditoriaDefault () *ReporteAuditoria {
    return &ReporteAuditoria {
        entidadBase : entidadBase {
            id: "RPT-BEF-C01",
            fechaCreacion: time.Now(),
        }
    }
}
```

```
NombreAuditor: "NN"
Sistema: "Sistema Generico",
PuntuacionSeguridad: 5.0
Completado: false
```

```
}
```

```
3
```

```
func NewReporteAuditoria (sistema string, auditor string)
    *ReporteAuditoria {
```

```
r := NewReporteAuditoriaDefault()
r.sistema = sistema
r.NombreAuditor = auditor
r.entidadBase.ID = fmt.Sprintf ("RPT-%d-%d", time.
Now () .Format ("0111"))
return r
```

```
3
```

```
func NewReporteAuditoriaFull (sistema string, auditor string,
puntuacion float64, id string, completado bool) *ReporteAuditoria,
```

```
r := ReporteAuditoria {
    entidadBase: entidadBase {
        id: id,
        fechaCreacion: time.Now (),
    }
}
```

```
3
```

Nombre Autor auditor,
Sistema : sistema
puntuacion, seguridad : puntuacion,
completado : completo,

3
return r

2
func NewReporteAuditoriaCopy (original * ReporteAuditoria)
* ReporteAuditoria f
Copiar := * original
Copiar. EntidadBase. FechaCreacion = time.Now()
Copiar. EntidadBase.ID = original.EntidadBase.ID + " (cp)"
return Copiar

3
type UsuarioAuditado struct {
ID Usuario string
Nombre string
Rol string
Activo bool

3
func NewUsuarioAuditadoDefault () * UsuarioAuditado {
return UsuarioAuditado {IDUsuario: "User - 000", Nombre:
"Invitado", Rol: "Visitante", Activo: false}}

3
func NewUsuarioAuditado (id string, nombre string) * UsuarioAuditado {
U := NewUsuarioAuditadoDefault ()
U.IDUsuario = id
U.Nombre = nombre
U.Activo = true
return U}

3
func NewUsuarioAuditadoFull (id string, nombre string, rol
string, activo bool) * UsuarioAuditado {
return UsuarioAuditado {IDUsuario: id, Nombre: nombre,
rol, Activo: activo}}

3
func NewUsuarioAuditadoCopy (original * UsuarioAuditado) * UsuarioAuditado {
Copiar := * original
Copiar.IDUsuario = original.IDUsuario + " - copy"
return Copiar}

3

type RequisitosSeguridad struct {

Codigo	String
Descripcion	String
Gravedad	Int
Establecimiento	Bool

3

```
func NewRequisitoSeguridadDefault() * RequisitoSeguridad {
    return &RequisitoSeguridad{Codigo: "REQ-000", Descripcion: "Requisito pendiente", Gravedad: 1, Establecimiento: false}
```

4

```
func NewRequisitoSeguridad(Codigo string, gravedad int) * RequisitoSeguridad {
```

```
r := NewRequisitoSeguridadDefault()
```

```
r.Codigo = Codigo
```

```
r.Gravedad = gravedad
```

```
r.Establecimiento = true
```

```
return r
```

5

```
func NewRequisitoSeguridadFull(Codigo string, Desc string, grav int, establecimiento bool) * RequisitoSeguridad {
    return &RequisitoSeguridad{Codigo: Codigo, Descripcion: Desc, Gravedad: grav, Establecimiento: establecimiento}}
```

6

```
func NewRequisitoSeguridadCopy(original * RequisitoSeguridad) * RequisitoSeguridad {
```

```
copy := *original
```

```
copy.Codigo = original.Codigo + " Copy"
```

```
return copy
```

7

type ProyectoPenetracion struct {

Plataforma	String
------------	--------

Metodologia	String
-------------	--------

TiempoEstimado	Time.Duration
----------------	---------------

VulnerabilidadesEncontradas	Int
-----------------------------	-----

8

```
func NewPruebaPenetracion Default () * PruebaPenetracion {
    return PruebaPenetracion { Plataforma: "Web", Metodologia: "OSSTIME", TiempoEstimado: 24, Time: Hour, VulnerabilidadesEncontradas: 0 }
}
```

```
func NewPruebaPenetracion (Plataforma string, TiempoTime Duration) * PruebaPenetracion {
}
```

```
return PruebaPenetracion { Plataforma: Plataforma, Metodologia: Metodo, TiempoEstimado: Tiempo, VulnerabilidadesEncontradas: VULS }
}
```

```
func NewPruebaPenetracion Copy (origina) * PruebaPenetracion {
    * PruebaPenetracion {
        Copia := origina
    }
    return Copia
}
```

```
type HallazgoFallo struct {
}
```

```
CodigoFallo struct {
    Severidad string
    ComponenteAfectado string
    FechaDeOcurrimiento time.Time
}
```

```
func NewHallazgoFallo Default () * HallazgoFallo {
    return HallazgoFallo { CodigoFallo: "fallo", Severidad: "Baja", ComponenteAfectado: "Pescador", FechaDeOcurrimiento: Now() }
}
```

```
func NewHallazgoFallo (Codigo string, Severidad string, Componente string, Fecha time.Time) * HallazgoFallo {
    return HallazgoFalloCopy (original * HallazgoFallo)
}
```

```
func NewHallazgoFallo (Codigo string, Severidad string) * HallazgoFallo {
}
```

```
h := NewHallazgoFallo Default ()

```

```
h.CodigoFallo = Codigo

```

```
h.Severidad = Severidad

```

```
return h
}
```

```
func NewultimoAlgoFullCopy (original *ultimoAlgo) {
```

```
    copia := original
```

```
    copia.Codigo = original.Codigo.FULL + "Copy"
```

```
    return copia
```

```
}
```

Estudiante

```
type Estudiante Accionable Interface {
```

```
    Estudiar (horas int)
```

```
    CompletarTema (tema String)
```

```
    MostrarEstado()
```

```
}
```

```
type Evaluador Interface {
```

```
    Evaluar (puntaje float64)
```

```
    Certificar ()
```

```
}
```

```
type EstadisticaBase Struct {
```

```
    fechaInicio time.Time
```

```
    DiasTotales int
```

```
    LenguajeBase string
```

```
}
```

```
func NewEstadisticaBase (lenguaje string) EstadisticaBase {
```

```
    return EstadisticaBase {
```

```
        fechaInicio: time.Now(),
```

```
        DiasTotales: 0,
```

```
        lenguajeBase: lenguaje,
```

```
}
```

```
}
```

```
type AprendiendoGO_Refactor Struct {
```

```
    EstadisticaBase
```

```
    NombreEstudiante string
```

```
    NivelActual string
```

```
    HorasEstudio int
```

```
    AvancePorcentaje float64
```

```
    Activo bool
```

```

    ProgresoSemanal | int
    temasCompletados | string
    calificacion | float64
    feedbackVprofesor | string
    configuracion | map[EString] interface {}

    Certificado bool
    UltimTermo string

func (a *AprendiendoGO) InicializarEstado() {
    a.NivelActual = "Principiante"
    a.Activo = true
    a.TemasCompletados = []string {}
    a.ProgresoSemanal = []int {}
    a.Configuracion = make(map[EString] interface{})
}

func NewAprendiendoGO() *AprendiendoGO {
    a := AprendiendoGO{
        EstadisticaBase: NewEstadisticaBase("GO"),
        NombreEstudiante: "Estudiante " + strconv.Itoa(rand.Intn(100)),
        HorasEstudio: 0,
        AvancePorcentaje: 0.0,
    }
    a.InicializarEstado()
    return a
}

func NewAprendiendoGOFull(nombre string, nivel string, horas
    int, lenguaje string, activo bool) *AprendiendoGO {
    a := AprendiendoGO{
        EstadisticaBase: NewEstadisticaBase(lenguaje),
        NombreEstudiante: nombre,
        NivelActual: nivel,
        HorasEstudio: horas,
        AvancePorcentaje: float64(horas) * 2.5,
        Activo: activo,
    }
    a.InicializarEstado()
    return a
}

```

func NuevoAprendiendo60 (Original & Aprendiendo60) * Aprendiendo60 {

Copia := NuevoAprendiendo60 {

Original. NombreEstudiante,

Original. NivelActual,

Original. HorasEstudio,

Original. EstadisticaBase -> LenguajeBase,

Original. Activo,

Copia.TemasCompletados = append ([] String { }), Original.TemasCompletados)

Copia.progresoDemanda = append ([] int { }), Original.progresoDemanda)

if Original.Configuracion != nil {

Copia.Configuracion = make [map [String] Interface { }]

for K, V := range Original.Configuracion {

Copia.Configuracion [K] = V

}

Copia.AvancePorcentaje = Original.AvancePorcentaje

Copia.Calificacion = Original.Calificacion

Copia.FeedbackProfesor = Original.FeedbackProfesor

Copia.Certificado = Original.Certificado

Copia.UltimoTema = Original.UltimoTema

Copia.PuntasTotal = Original.PuntasTotal

Copia.FechaInicio = Original.FechaInicio

return Copia

}

func (a * Aprendiendo60) Estudiar (horas int) {

if ! a.Activo {

fmt.Println ("No estudiante activo")

return

}

a.HorasEstudio += horas

a.AvancePorcentaje += float64(horas) * 2.5

if a.AvancePorcentaje >= 100 {

a.AvancePorcentaje = 100

}

a. Dias totales
fmt. printf ("%.3f estudió %.d horas. Progreso: %.1f %.1%\n",
a. NombreEstudiante, horas, a. AvancePorcentaje)

}

func (a *AprendiendoGO) CompletarTema (tema string) {

a. TemasCompletados = append (a.TemasCompletados, tema)

a. UltimoTema = tema

fmt. printf ("Tema Completado: %.3s\n", tema)

if a. AvancePorcentaje >= 25 a. NivelActual = "Básico"
a. NivelActual = "Intermedio"

}

}

func (a *AprendiendoGO) Evaluar (puntaje float64) {

a. Calificación = puntaje

if puntaje >= 3.5 {

a. FeedbackProfesor = "Buen desempeño,"

}

a. FeedbackProfesor = "Reforzar"

}

fmt. printf ("EV Completado. Nota: %.1f | Feedback: %.3s\n",
puntaje, a. feedbackProfesor)

}

func (a *AprendiendoGO) Certificar () {

if a. AvancePorcentaje >= 100 {

a. Certificado = true

fmt. printf ("Felicitación! %s hasta Completar Curso %.3s\n",

a. NombreEstudiante, a. LenguajeBase)

}

else {

fmt. printf ("No alcanza el 100% del proceso %.3s\n",
a. LenguajeBase)

}

}

punt (a *AprendiendoGO) MostrarFeedback() {

fmt. Println ("ESTÁNDAR PRENDIZAJE")

fmt. Println ("Estudiante: %.3s\nNivel: %.3s\nProgreso:

%.1f %.1%\nHoras: %.3d\n",

a. NombreEstudiante, a. NivelActual, a. AvancePorcentaje, a. HorasEstudio)

fmt. Println ("Tematica: %.3s\nCertificado: %.1t\n",

```
a - Ultimo Tengo, a - Certificado  
int printf ("Iniciodc : %s (Dias totales : %d, Lenguaje : %s)  
\n", a - fechaInicio.format ("u"), a - diasTotales, a - lenguajeBase)
```

```
func ( a * Aprendiendo GO ) SetConfiguracion ( config map [ ] string )  
interface {}  
a - configuracion = config
```

```
func ( a * Aprendiendo GO ) MostrarConfiguracion () {
```

```
if len ( a - configuracion ) == 0 {  
fmt.Println (" Sin Configuracion adicionales")  
return
```

```
data _ := json.MarshalIndent ( a - configuracion, "", " ")  
fmt.Println ("Configuracion actual : ")  
fmt.Println ( string ( data ))
```