

JAU

```
Public boolean isAcerteCorrecto () {  
    return acerteCorrecto;  
}  
  
Public void setAcerteCorrecto (boolean acerteCorrecto) {  
    this.acerteCorrecto = acerteCorrecto;  
}  
  
Public JSONObject GetSensores () {  
    Returns Sensores;  
}  
  
Public void SetSensores (JSONObject Sensores) {  
    this.Sensores = Sensores;  
}  
  
- Método de acción  
  
Public void encender () {  
    If (acerteCorrecto) {  
        funcionamiento = True;  
        temperatura = 10;  
        System.out.println ("Motor encendido (Correctamente);  
    }  
    else {  
        System.out.println ("No se puede encender, Nivel bajo de  
        acierto");  
    }  
}  
  
Public void apagar () {  
    funcionamiento = false;  
    temperatura = 25.0;  
    System.out.println ("Motor apagado y temperatura establecendo");  
}  
  
Public void mostrarInfo () {  
    System.out.println ("Información del Motor");  
    System.out.println ("Tipo de Motor: " + tipoMotor);  
    System.out.println ("Cilindros: " + cilindros);  
    System.out.println ("Potencia " + Potencia + " ");  
}
```

```
System.out.println("Temperatura actual: " + temperatura + " °C");
System.out.println("Aceite en buen estado: " + aceiteCorrecto);
```

{

}

• Sistema de frenado

(3)

```
public class SistemaFrenado {
```

```
    public String tipoFreno;
    public boolean frenoActivo;
    public int presionActual;
    public String fabricante;
    public double distanciaDetencion;
```

```
    private boolean nivelLiquidoCorrecto;
    private double temperaturaPastilla;
    private UDNOBJECT sensores;
    private byte codigoError;
    private String numeroSerie;
```

```
    protected double desgastePastillas;
    protected boolean necesitaCambio;
    protected String tiroLiquido;
    protected int presionMaxima;
    protected String paisFabricacion;
```

```
    this.tipoFreno = tipoFreno;
```

```
    this.frenoActivo = frenoActivo;
```

```
    this.fabricante = fabricante;
```

```
    this.distanciaDetencion = distanciaDetencion;
```

```
    this.presionActual = presionActual;
```

```
    this.frenoActivo = frenoActivo;
```

```
    this.nivelLiquidoCorrecto = true;
```

```
    this.desgastePastillas = 10.5;
```

```
    this.temperaturaPastilla = 30.0;
```

```
    public boolean isNivelLiquidoCorrecto() {
```

```
        return nivelLiquidoCorrecto;
```

}

```
    public void setNivelLiquidoCorrecto(boolean nivelLiquidoCorrecto) {
```

```
        this.nivelLiquidoCorrecto = nivelLiquidoCorrecto;
```

}

```
    public double getTemperaturaPastilla() {
```

```
        return temperaturaPastilla;
```

}

Public void apagar () {

this. encendido = falso;
System.out.println ("Carro apagado")

}

3

• Partes del motor de Un Carro

Package mixto;

Import org.json.JSONObject;

Public Class motor {

Public String tipoMotor;

Public int cilindro;

Public double potencia;

Public boolean funcionamiento;

Public String fabricante;

Private double temperatura;

Private boolean aceiteCorrecto;

Private byte codigoDiagnosticos;

Private JSONObject sensores;

Private String numeroSerie;

Protected revolucionesPorMinuto;

Protected consumoCombustible;

Protected tipCombustible;

Protected necesitaMantenimiento;

Protected paraFabricante;

- Constructor

Public Motor (String tipoMotor; int cilindro; double potencia; boolean

funcionamiento; String fabricante; boolean funcionamiento) {

this. tipoMotor = tipoMotor;

this. cilindro = cilindro;

this. potencia = potencia;

this. fabricante = fabricante;

this. funcionamiento = funcionamiento;

this. temperatura = 25.0 inicial;

this. aceiteCorrecto = true;

3 Public double GetTemperatura () {

return temperatura;

3 Public void SetTemperatura (double temperatura) {

this. temperatura = temperatura;

3

2

1 Clases

```
1
• Public class Coche {
    Public String marca;
    public String modelo;
    public double precio;
    public int puertas;
    public boolean disponible;

    Private String color;
    private boolean encendido;
    private char tipoCombustible;
    private byte codigoChasis;
    private JSONObject datosTecnicos;

    Protected String tipotransmision;
    Protected int añoFabricacion;
    Protected double kilometraje;
    Protected boolean mantenimiento;
    Protected String paisOrigen;

    public Coche (String marca, String modelo, double precio, int puertas,
    boolean disponible) {
        this.marca = marca;
        this.modelo = modelo;
        this.precio = precio;
        this.puertas = puertas;
        this.disponible = disponible;
        this.encendido = false;
    }
```

```
3
    Public boolean Encendido () {
        Return isEncendido;
    }
```

```
3
    Public void setEncendido ('boolean encendido) {
        This.encendido = encendido;
    }
```

```
3
    Public JSONObject Get DatosTecnicos () {
        Return datosTecnicos;
    }
```

```
3
    Metodo de accion
```

```
1
    Public void encender () {
        This.encendido = True;
        System.out.println ("Coche encendido");
    }
```

• GET, SET / Privados

```

public void setTemperaturaPastillas (double temperaturaPastillas) {
    this.temperaturaPastillas = temperaturaPastillas;
}

public JSONObject getSensores () {
    return Sensores;
}

public void setSensores (JSONObject Sensores) {
    this.Sensores = Sensores;
}

public void aplicarFreno () {
    if (NivelLiquidoCorrecto < desgastePastillas) {
        FrenoActivo = True;
        presionActual = 10;
        distanciaDetencion = 1.5;
        temperaturaPastillas = 10;
        System.out.println ("freno aplicado Correctamente, Vehiculo desacelerando");
    } else {
        System.out.println ("No se puede frenar Verificar NivelLiquido, cambia pastillas");
    }
}

public void liberarFreno () {
    FrenoActivo = False;
    presionActual = 10;
    temperaturaPastillas = 15;
    System.out.println ("freno liberado Vuelve a su estado Normal");
}

public void mostrarEstado () {
    System.out.println ("Estado del Sistema de frenado");
    System.out.println ("tipo de freno " + tipoFreno);
    System.out.println ("fabricante : " + fabricante);
    System.out.println ("temperatura de pastillas : " + temperaturaPastillas);
    System.out.println ("presionactual " + presionActual);
    System.out.println ("desgaste de pastillas : " + desgastePastillas);
    System.out.println ("nivel de liquido Correcto : " + nivelLiquidoCorrecto);
}

```

• Método de Pago en Línea (4)

```
class MetodoDePagoEnLinea {
```

```
    public String nombreTitular;  
    public String tipoPago;  
    public double monto;  
    public boolean transaccionExitsa;  
    public int numeroTransaccion;
```

```
    private String numeroTarjeta;  
    private String codigoSeguridad;  
    private boolean Verificacion;  
    private byte comprobanteBinario;  
    private JSONObject detallePago;
```

```
    protected String paisOrigen;  
    protected double tasaImpuesto;  
    protected boolean requiereFactura;  
    protected String MetodoMoneda;  
    protected int tiempoProcesamiento;
```

```
    public MetodoDePagoEnLinea(String nombreTitular, String tipoPago,  
        double monto, int numeroTransaccion, boolean transaccionExitsa) {
```

```
        this.nombreTitular = nombreTitular;
```

```
        this.tipoPago = tipoPago;
```

```
        this.monto = monto;
```

```
        this.numeroTransaccionExitsa = transaccionExitsa;
```

```
        this.Verificacion = false;
```

```
        this.tasaImpuesto = 0.19;
```

```
        this.tiempoProcesamiento = 0;
```

```
    public boolean IsVerificacion () {
```

```
        return Verificacion;
```

```
}
```

```
    public void SetVerificacion (boolean Verificacion) {
```

```
        this.Verificacion = Verificacion;
```

```
}
```

```
    public JSONObject GetDetallePago () {
```

```
        return detallePago;
```

```
}
```

```
    public void SetDetallePago (JSONObject detallePago) {
```

```
        this.detallePago = detallePago;
```

```
}
```

```
Public void procesarpago () {  
    if (Verificacion == monto > 0) {  
        transaccionExito = true;  
        tiempoProcesamiento = 5;  
        System.out.println ("Pago exitoso por " + nombreTitular);  
        System.out.println ("Monto total: $" + monto);  
    }  
}
```

else {

transaccionExito = false;
System.out.println ("No se puede proceder el pago. Verifica el monto
la autenticación.");

```
}  
}  
}
```

```
Public void generarComprobante () {
```

```
if (transaccionExito) {  
    System.out.println ("Comprobante generado para la transacción #1" + numero  
    transaccion);
```

```
}  
else {  
    System.out.println ("No se puede generar Comprobante al pago no  
    fue exitoso");
```

```
Public void mostrarResumen () {
```

```
System.out.println ("Resumen del pago en línea");  
System.out.println ("titular: " + nombreTitular);  
System.out.println ("tipo de pago: " + tipoPago);  
System.out.println ("Monto: $" + monto);  
System.out.println ("transacción exitosa: " + transaccionExito);  
System.out.println ("Impuesta aplicada: " + tasaImpuesta * 100 + "%");  
System.out.println ("país de origen: " + paisOrigen);
```

Control Embarazo

(5)

```
Public String nombreMadre;
Public int edadMadre;
Public double pesoActual;
Public int SemanaGestacion;
Public boolean riesgoAlto;

Private string tipoSangre;
Private double presionArterial;
Private boolean VitaminasTomadas;
Private byte EcografiaKlinica;
Private UDONObject historialClinico;

Protected string MedicoEncargado;
Protected string hospitalReferencia;
Protected boolean SeguimientoActivo;
Protected int ControlesRealizados;
Protected double nivelHemoglobina;
```

```
public ControlEmbarazo (String nombreMadre, int edadMadre, double pesoActual,
int SemanaGestacion, boolean riesgoAlto) {
```

```
this.nombreMadre = nombreMadre;
```

```
this.edadMadre = edadMadre;
```

```
this.pesoActual = pesoActual;
```

```
this.SemanaGestacion = SemanaGestacion;
```

```
this.riesgoAlto = riesgoAlto;
```

```
this.VitaminasTomadas = false;
```

```
this.SeguimientoActivo = true;
```

```
this.ControlesRealizados = 0;
```

```
}
```

```
public boolean IsVitaminasTomadas () {
```

```
return VitaminasTomadas;
```

```
}
```

```
public void SetVitaminasTomadas (boolean VitaminasTomadas) {
```

```
this.VitaminasTomadas = VitaminasTomadas;
```

```
}
```

```
public UDONObject GetHistorialClinico () {
```

```
return historialClinico;
```

```
}
```

```
public void registrarControl (double nuevoPeso, double presion, double hemoglobina) {
```

```
this.pesoActual = nuevoPeso;
```

```
this.presionArterial = presion;
```

```
this.nivelHemoglobina = hemoglobina;
```

```
this.controlRealizado ++;
```

```
System.out.println ("Control registrado Correctamente Total de Controles " +
```

```
controlRealizado);
```

```
}
```

```
public void mostrarResumen () {  
    System.out.println ("Resumen del control de embarazo");  
    System.out.println ("Madre: " + nombreMadre + " (edad: " + edadMadre + ")");  
    System.out.println ("Semanas de gestación: " + SemanasGestacion);  
    System.out.println ("Peso actual: " + pesoActual + " kg");  
    System.out.println ("Embarazo de alto riesgo: " + (riesgoAlto ? "Sí" : "No"));  
    System.out.println ("Controles realizados " + controlesRealizados);  
    System.out.println ("Nivel de hemoglobina: " + nivelHemoglobina);  
}
```

```
}  
  
public void evaluarRiesgo () {  
    if (presionArterial > 140 || nivelHemoglobina < 11) {  
        riesgoAlto = true;  
        System.out.println ("Riesgo alto detectado. Se requiere control médico urgente");  
    } else {  
        riesgoAlto = false;  
        System.out.println ("Estado estable. Continuar controles normales");  
    }  
}  
}
```

JAVA 8 CRIST

Clase ControlTiempo Ejercicio {

 nombre Usuario; - Publicas

 tipo Ejercicio;

 duracion Minutos;

 calorias Quemadas;

 Sesion Activa;

 # Frecuencia Cardiaca; - Privadas

 # nivel hidratacion;

 # RangoBinario;

 # detalles JDON;

 # autenticado;

 - FechaInicio;

 ObjetivoMinutos;

 Intensidad;

 descansoRecomendado;

 Comentarios;

 • Constructor (nombreUsuario, tipoEjercicio, duracionMinutos,
 caloriasQuemadas, SesionActiva) {

 this.nombreUsuario = nombreUsuario;

 // Valores por defecto

 this.tipoEjercicio = tipoEjercicio;

 this.duracionMinutos = duracionMinutos;

 this.caloriasQuemadas = caloriasQuemadas;

 this.SesionActiva = SesionActiva;

 this. # FrecuenciaCardiaca = 0;

 this. # nivelHidratacion = 100;

 this. Intensidad = "Medio";

 this. descansoRecomendado = 10;

 this. Comentarios = " ";

}

 get FrecuenciaCardiaca 0 {

 return this. # FrecuenciaCardiaca;

}

 Set FrecuenciaCardiaca (valor) {

 if (valor > 0 && Valor < 220) {

 this. # FrecuenciaCardiaca = Valor;

 } else {

 Console.Warn(" Frecuencia Cardiaca no Valida);

}

}

get detallesJSON (Obj) {

this. # detallesJSON = Obj;

}

- Metodos de accion

IniciarSesion () {

this. # autenticado = true;

this. fechaInicio = New Date ();

this. sesionActiva = true;

console.log ('* Sesión iniciada por ' + { this.nombreUsuario } al '');

this. - fechaInicia. toLocaleString ('Z');

}

Registrar Ritmo (frecuencia, hidratacion) {

this. frecuenciaCardiaca = frecuencia;

this. # nivelHidratacion = hidratacion;

console.log ('Ritmo Cardiaco: ' + { frecuencia } bpm | hidratacion: ' + { hidratacion } %');

}

FinalizarSesion () {

if (!this.sesionActiva) {

console.log ('* No hay sesión activa para finalizar');
return;

}

This. sesionActiva = false;

const tiempoReal = Math.floor (Math.random () * this.duracionMinutos);

console.log ('* Sesión finalizada. duración real: ' + { tiempoReal } min.

calorías = ' + { this.caloriasQuemadas });

}

mostrarResumen () {

console.log ('* Resumen de la Sesión');

console.log ('* Usuario: ' + { this.nombreUsuario });

console.log ('* ejercicio: ' + { this.tipoEjercicio });

console.log ('* Duración planificada: ' + { this.duracionMinutos } minutos');

console.log ('* Calorías quemadas: ' + { this.caloriasQuemadas });

console.log ('* Frecuencia Cardiaca: ' + { this. # frecuenciaCardiaca } bpm);

console.log ('* Nivel de hidratación: ' + { this. # nivelHidratacion } %);

console.log ('* Intensidad: ' + { this. intensidad });

}

Recomendar Deseando () {

IF (this. # frecuenciaCardiaca > 160) {

this. - deseoRecomendado = 15;

console.log ('* Deseando recomendado: 15 minutos (frecuencia alta)');

} else {

Config lag (1 descanso recomendado: 82 this. - descanso Recomendado 3 minutos);

3;

3;

3;

Sesion Sesion 1 = new ControlTiempo Ejercicio ("Carlos Lopez", "Ciclismo", 45, 350, false);

Sesion 1. datosJSON = {

Lugar: "Gimnasio Central",

Musica: "Pop Energetico",

Fecha: "2025-11-09"

};

Inicio Sesion

Sesion 1. IniciarSesion();

Registro de Ritmo

Sesion 1. registrarRitmo (155, 85);

Mostrar resumen.

Sesion 1. mostrarResumen();

Recomendacion descanso

Sesion 1. recomendarDescanso();

Finalizar

Sesion 1. finalizarSesion();

• Control celular

Class ControlCelular {

marca,

modelo,

sistema operativo,

almacenamiento,

encendido;

#nivelBateria = 100;

#conectadoWifi = false;

#codigoDesbloqueo = "0000";

#configuracionJSON = null;

```

- VersionSoftware
- FechaUltimaCarga
- modoOscuro
- temperatura =
- ListaAplicaciones = ("telefono", "Mensaje", "Camara", "Galeria")
}

Constructor (marca, modelo, sistemaOperativo, almacenamiento, encendido=false) {
    this.marca = marca;
    this.modelo = modelo;
    this.sistemaOperativo = sistemaOperativo;
    this.almacenamiento = almacenamiento;
    this.encendido = encendido;
}

get nivelBateria () {
    return this.#nivelBateria;
}

set nivelBateria (nivel) {
    if (nivel < 0 || nivel > 100) {
        Consola.warn("nivel de bateria debe estar entre 0 y 100");
        return;
    }
    this.#nivelBateria = nivel;
}

get ConfiguracionJSON () {
    return this.#ConfiguracionJSON;
}

set ConfiguracionJSON (config) {
    this.#ConfiguracionJSON = config;
}

encenderCelular () {
    if (!this.encendido) {
        Consola.log("al encender esta encendido.");
        return;
    }
}

this.encendido = true;
Consola.log(`{this.marca} {this.modelo} ${this.encendido}`);
}

apagarCelular () {
    if (!this.encendido) {
        Consola.log("al apagar ya esta apagado");
        return;
    }
}

```

```
this. encendido = false;
console.log ("Calcular apagado Correctamente.");
}
```

```
CargaBateria () {
this. fechaUltimaCarga = new Date (1);
this. #nivelBateria = 100;
Console.log ("Bateria al 100% ultima Carga " + this. -fechaUltimaCarga,
talocaleString (13));
}
```

```
Conectar Wifi (red) {
this. #conectadoWifi = true;
Console.log ("Conectado a " + red);
}
```

```
abrir Application (app) {
if (!this. encendido) {
console.log ("encendido el celular Primero.");
return;
}
if (!this. -ListaAplicaciones. Incluido (app)) {
Console.log (" " + app + " no esta instalado.");
return;
}
}
```

```
Console.log ("abriendo " + app);
this. #nivelBateria = Math. max (0, this. #nivelBateria - 3);
}
```

```
Instalar Application (app) {
if (!this. -ListaAplicaciones. Include (app)) {
console.log (" " + app + " ya esta instalado.");
return;
}
}
```

```
this. -ListaAplicaciones. push (app);
Console.log (" " + app + " instalado Correctamente.");
}
```

```
Mostrar Estado () {
Console. log (" Estado del celular ");
Console. log (" Marca: " + this. marca);
Console. log (" Modelo: " + this. modelo);
Console. log (" SO: " + this. sistemaOperativo + " " + this. VersionSoftware);
Console. log (" almacenamiento: " + this. almacenamiento + " GB");
Console. log (" encendido: " + this. encendido ? "SI" : "NO");
Console. log (" Bateria: " + this. #nivelBateria + "%");
Console. log (" modo Oscuro: " + this. -modoOscuro ? "activo" : "desactivado");
Console. log (" Apps: " + this. -ListaAplicaciones. LstApp ());
}
}
```

```
Const miCelular = new ControCelular ("Samsung", "Galaxy S24",  
"android", 256);  
miCelular.ConfiguracionJSON = {  
    idioma: "Español",  
    region: "Latinoamericano",  
    brillo: 70,  
    sonido: true  
};
```

} ;

```
miCelular.encenderCelular();  
miCelular.conectarWifi("red casa_56");  
miCelular.abrirAplicacion("camara");  
miCelular.instalarAplicacion("whatsapp");  
miCelular.mostrarFoto();  
miCelular.CargarBateria();  
miCelular.apagarCelular();
```

```

class ControlLamparas {
    marca;
    color;
    tipo;
    ubicacion;
    encendida;
    #intensidad = 50;
    #horasUso = 0;
    #temperaturaColor 400;
    #ConsumoWatts = 10;
    #Configuracion = null;

    -VidaUtil = 1000;
    -fechaInstalacion = null;
    -modoAhorro = false;
    -tempActual = 25;
    estadosGuardados = [];

    constructor (marca, color, tipo, ubicacion, encendida = false) {
        this.marca = marca;
        this.color = color;
        this.tipo = tipo;
        this.Ubicacion = ubicacion;
        this.encendida = encendida;
        this.fechaInstalacion = new Date();
    }

    get Intensidad () {
        return this.#intensidad;
    }

    set Intensidad (valor) {
        if (valor < 0 || valor > 100) {
            console.warn(`Intensidad debe estar entre 0 y 100`);
        }
        this.#Intensidad = valor;
    }

    get Configuracion () {
        return this.#Configuracion;
    }

    set Configuracion (config) {
        this.#Configuracion = config;
    }

    encender () {
        if (!this.encendida) {
            console.log(`Lampara ya esta encendida.`);
        }
        return;
    }
}

```

this. encendida = True
Console.log ('La lámpara ' + this.marca) encendida en ' + this.localización
}; ');

} apagar () {
if (!this.encendida) {
Console.log ('La lámpara ya está apagada.');
return;

} this. encendida = false;
Console.log ('La lámpara apagada.');

} ajustar intensidad (valor) {
if (!this. encendida) {
Console.log ('Enciende la lámpara primero.');
return;

} this.intensidad = valor;
Console.log ('intensidad ajustada a ' + this.intensidad + ' %');

Cambiar temperatura (kelvin) {
if (kelvin < 2700 || kelvin > 6500) {
Console.log ('Temperatura entre 2700K y 6500K');
return;

} activar modo Ahorro () {
this. modoAhorro = true;
this. #intensidad = 30;
this. #consumoWatts = 3;
Console.log ('modo ahorro');

} Guardar Estado () {
const estado = {
intensidad: this. #intensidad,
temperatura: this. #temperaturacolors,
fecha: new Date ()

};
this.estadosGuardados.push (estado);
Console.log ('Estado guardado correctamente.');

}

```

class Cafetera {
    marca;
    modelo;
    capacidad;
    material;
    encendida;
    nivelAgua = 0;
    nivelCafe = 0;
    temperatura = 20;
    preparando = false;
    configuracion = null;

    tipoMolienda = "Medio";
    intensidadCafe = "Normal";
    tazasPreparadas = 0;
    ultimoUso = null;
    recetasGuardadas = [];

    constructor (marca, modelo, capacidad, material, encendida = false) {
        this.marca = marca;
        this.modelo = modelo;
        this.capacidad = capacidad;
        this.material = material;
        this.encendida = encendida;
    }

    get nivelAgua () {
        return this.nivelAgua;
    }

    set nivelAgua (cantidad) {
        if (cantidad < 0 || cantidad > this.capacidad) {
            console.warn ("agua entre 0 y " + this.capacidad + " ml");
        }
        this.nivelAgua = cantidad;
    }

    get Configuracion () {
        return this.configuracion;
    }

    set Configuracion (config) {
        this.Configuracion = config;
    }

    encender () {
        if (!this.encendida) {
            console.log ("Cafetera ya esta encendida.");
        }
        return;
    }
}

```

```

this.encendida = true;
console.log("Cafetera " + {this.marca} + " " + {this.modelo} + " encendido.");
}

apagar () {
  if (!this.encendida) {
    console.log("Cafetera ya esta apagada.");
    return;
  }
  if (this.preparando) {
    console.log("No puedes apagar mientras prepara cafe.");
    return;
  }
  this.encendida = false;
  console.log("Cafetera apagada.");
}

llenarAgua (Cantidad) {
  const total = this.nivelAgua + Cantidad;
  if (total > this.Capacidad) {
    console.log("Capacidad maxima: " + {this.Capacidad} + " ml");
    return;
  }
  this.nivelAgua = total;
  console.log('Agua agregada: Nivel actual: ' + {this.nivelAgua} + " ml");
}

agregarCafe (tazas = 1) {
  if (!this.nivelCafe + tazas * 150) {
    console.log("Cafe agregado: " + {tazas * 150} + " g. Total: " + {this.nivelCafe} + " g");
  }
}

PrepararCafe (gramos) {
  this.nivelCafe += gramos;
  console.log("Enciende la cafetera primero.");
  return;
}

const aguaNecesaria = tazas * 150;
const cafeNecesario = tazas * 10;

if (!this.nivelAgua < aguaNecesaria) {
  console.log("No hay suficiente agua.");
  return;
}

if (!this.nivelCafe < cafeNecesario) {
  console.log("No hay suficiente cafe.");
  return;
}
}

```

```
    Registrar horas Uso (horas) {
        this. # horaUso += horas;
        const vidaRestante = this.vidaUtil - this. # horaUso;
        console.log (" { horas } h registradas. Vida restante: { vidaRestante } h ");
    }
```

```
3 mostrarEstado () {
```

```
    console.log (" Estado de la lampara ");
    console.log (" Marca: { this.marca } ");
    console.log (" Color: { this.color } ");
    console.log (" tipo: { this.tipo } ");
    console.log (" Ubicacion: { this.ubicacion } ");
    console.log (" encendido: { this.encienda ? "SI" : "NO" } ");
    console.log (" Intensidad: { this.intensidad } ");
    console.log (" temperatura: { this.temperaturaColor } K ");
    console.log (" consumo: { this.consumoWatt } W ");
    console.log (" horas de uso: { this.horaUso } h ");
    console.log (" Modo ahorro: { this. # modoAhorro ? "Activo" : "Preactivo" } ");
}
```

```
3;
```

```
3 const lamparaSala = new ControlLampara ("Philips", "Blanco", 100, "Sala");
```

```
LamparaSala.Configuracion = {
    programacion: "Automatica",
    sensor: "movimiento",
    conectividad: "WiFi"
};
```

```
3;
```

```
LamparaSala.enciender ();
LamparaSala.ajustarIntensidad (80);
LamparaSala.CambiarTemperaturaColor (3000);
LamparaSala.guardarEstado ();
LamparaSala.mostrarEstado ();
LamparaSala.activarModoAhorro ();
LamparaSala.registrarHorasUso (5);
LamparaSala.apagar ();
```

```

this.encendida = true;
console.log("Cafetera " + this.marca + " " + this.modelo + " encendido.");
}

apagar () {
  if (!this.encendida) {
    console.log("Cafetera ya está apagada.");
    return;
  }
  if (this.preparando) {
    console.log("No puedes apagar mientras prepara café.");
    return;
  }
  this.encendida = false;
  console.log("Cafetera apagada.");
}

llenarAgua (Cantidad) {
  const total = this.nivelAgua + Cantidad;
  if (total > this.capacidad) {
    console.log("Capacidad máxima: " + this.Capacidad + " ml");
    return;
  }
  this.nivelAgua = total;
  console.log("Agua agregada: " + Cantidad + " ml");
}

agregarCafe (tazas = 1) {
  if (!this.nivelCafe + tazas * 150) {
    console.log("Café agregado: " + tazas * 150 + " g. Total: " + this.nivelCafe);
  }
}

prepararCafe (gramos) {
  this.nivelCafe += gramos;
  console.log("Enciende la cafetera primero.");
  return;
}

const aguaNecesaria = tazas * 150;
const cafeNecesario = tazas * 10;

if (this.nivelAgua < aguaNecesaria) {
  console.log("No hay suficiente agua.");
  return;
}

if (this.nivelCafe < cafeNecesario) {
  console.log("No hay suficiente café.");
  return;
}

```

```
this. preparando = true;
console.log ('Preparando la taza de café');
this.temperatura = 85;
this.nivelAgua = aguaNecesaria;
this.nivelCafe = CafeNecesario;
this.tazasPreparadas += 1;
this.ultimoUso = New Date();
```

```
settimeout () => {
    this.preparando = false;
    console.log ('Café listo.');
}
```

```
}, 2000);
```

```
}
```

```
ajustar Intensidad (intensidad) {
    const niveles = ["Suave", "Normal", "Fuerte"];
    if (niveles.includes (intensidad)) {
        console.log ('Intensidad debe ser Suave, normal, fuerte');
        return;
    }
}
```

```
this.intensidadCafe = intensidad;
console.log ('Intensidad: ' + intensidad);
```

```
}
```

```
guardarReceta (nombre, taza, intensidad) {
    const receta = { nombre, tazas, intensidad, fecha: New Date() };
    this.recetasGuardadas.push (receta);
    console.log ('Receta ' + nombre + ' guardada.');
}
```

```
}
```

```
limpiar () {
    if (!this.encendido) {
        console.log ('No se puede limpiar la cafetera sin encenderla');
        return;
    }
}
```

```
this.nivelAgua = 0;
this.nivelCafe = 0;
this.temperatura = 20;
console.log ('Cafetera limpia y lista para usar');
```

```
}
```

```
mostrarEstado () {
    console.log ('Estado de la cafetera');
    console.log ('Marca: ' + this.marca + ', Modelo: ' + this.modelo);
    console.log ('Capacidad: ' + this.capacidad + ' ml');
    console.log ('Material: ' + this.material);
    console.log ('Encendido: ' + this.encendido ? 'Sí' : 'No');
    console.log ('Nivel agua: ' + this.nivelAgua + ' ml');
    console.log ('Nivel cafe: ' + this.nivelCafe + ' ml');
    console.log ('Temperatura: ' + this.temperatura + ' °C');
    console.log ('Intensidad: ' + this.intensidad);
    console.log ('Preparando: ' + (this.preparando ? 'Sí' : 'No'));
}
```

```
}
```

```
Ronot Micafetera = New ControlCafetera ("Largo", "Corto", "100");  
micafetera.Configuracion = {  
    temporizador "06:00 AM",  
    apagadoAuto : true,  
    Idioma : "Español",  
};
```

```
micafetera. encender();  
micafetera. LlegarAgua(500);  
micafetera. AgregarCafe(50);  
micafetera. ajustarIntensidad("Fuerte");  
micafetera. guardarReceta();  
micafetera. MostrarEstado();  
micafetera. prepararCafe();
```

```
Set timeout () => {
```

```
    micafetera. apagar();  
    micafetera. Limpiar();
```

```
};
```

* Control Reloj

```
Class ControlReloj {  
    marca;  
    tipo;  
    color;  
    material;  
    funcionando;  
  
    hora = 12;  
    minutos = 0;  
    alarma = null;  
    bateria = 100;  
    Configuracion = null;  
  
    Formato24h = true;  
    columnaGuardados = [-1];  
    Cronometro = 0;  
    Luz = false;  
    SonidoAlarma = "Beep";  
  
    Constructor (marca, tipo, color, material) {  
        this.marca = marca;  
        this.tipo = tipo;  
        this.color = color;  
        this.material = material;  
        this.funcionando = true;  
    }  
  
    get bateria () {  
        return this.bateria;  
    }  
  
    set bateria (nivel) {  
        if (nivel < 0 || nivel > 100) {  
            console.log ("Bateria debe estar entre 0 y 100");  
        }  
        this.bateria = nivel;  
    }  
  
    get Configuracion () {  
        return this.Configuracion;  
    }  
  
    set Configuracion (config) {  
        this.Configuracion = config;  
    }  
  
    ajustarHora (hora, Minutos) {  
        if (hora < 0 || hora > 23 || Minutos < 0 || Minutos > 59) {  
            console.log ("Hora o minutos invalidos");  
        }  
        this.hora = hora;  
        this.Minutos = Minutos;  
        console.log ("Hora ajustada");  
        this.mostrarHora ();  
    }  
}
```

```

mostrarHora () {
    const h = this.formato24h ? this.hora : this.hora / 12 * 12;
    const m = this.minutos.toString().padStart(2, '0');
    const periodo = this.formato24h ? (this.hora >= 12 ? 'PM' : 'AM') : '';
    return `${h} ${m} ${periodo}`;
}

```

```
{
    ConfiguracionAlarma (hora, Minutos) {

```

```

        this.alarmas = { hora, Minutos };
        this.alarmasGuardadas.push({ hora, Minutos, activa: true });
        console.log(`Alarma configurada para las ${hora} ${Minutos}`);
        return `Alarma configurada para las ${hora} ${Minutos}`;
    }
}
```

```
{
    activarLuz {
        this.luz = this.luz;
        this.bateria = Math.max(0, this.bateria - 2);
        console.log(`Luz ${this.luz} ${this.luz ? "encendida" : "apagada"}`);
    }
}
```

```
{
    cambiarFormato () {
        this.formato = this.formato24h ? "24 horas" : "12 horas (AM/PM)";
        console.log(`Formato ${this.formato}`);
    }
}
```

```
{
    iniciarCronometro () {
        this.cronometro = 0;
        console.log(`Cronometro iniciado`);
        const intervalo = setInterval(() => {
            this.cronometro += 1;
            if (this.cronometro >= 10) {
                clearInterval(intervalo);
                console.log(`Cronometro detenido ${this.cronometro}`);
            }
        }, 1000);
    }
}
```

```
, 1000);
}
```

```
{
    mostrarEstado () {
        console.log(`Estado del reloj`);
        console.log(`Marca ${this.marca}`);
        console.log(`Tipo ${this.tipo}`);
        console.log(`Bateria ${this.bateria}`);
        console.log(`Luz ${this.luz ? "encendida" : "apagada"}`);
        console.log(`Alarma ${this.alarmasGuardadas}`);
    }
}
```

```
{
    miReloj = new ControlReloj ("Casio", "Digital", "Negro", "Plastico");
}
```

mi Reloj - Configuracion = {

20 nd
Idioma : "6 M 1 - 5"
3 ;
"Español"

mi Reloj - ajustarhora (14, 30);
mi Reloj - Configuracion Alarma (1, 0);
mi Reloj - activar Luz (1);
mi Reloj - CambiaFormato();
mi Reloj - MostrarEstado();
mi Reloj - IniciarChronometro();

C#

• Televisor

①

Using System
namespace Mixto

{

Class Televisor

{

- Propiedades

```
public string Marca { get; set; }
public string Modelo { get; set; }
public double pulgadas { get; set; }
public string tipoPantalla { get; set; }
public bool encendido { get; set; }

private int CanalActual;
private int Volumen;
private bool modoSilencio;
private List<int> canalesFavoritos;
private string configuracionImagen;

protected string SistemaOperativo;
protected string resolucion;
protected bool modoAhorroEnergia;
protected bool ConexionWifi;
protected string UltimaAppAbierta;

public ControlTelevisor(string marca, string modelo, double pulgadas,
String tipoPantalla)
```

marca = marca;

modelo = modelo;

pulgadas = pulgadas;

tipoPantalla = tipoPantalla;

encendido = false;

CanalActual = 1;

Volumen = 10;

modoSilencio = false;

canalesFavoritos = new List<int>();

configuracionImagen = "estandar";

sistemaOperativo = "Windows";

resolucion = "4K";

modoAhorroEnergia = false;

conexionWifi = false;

ultimaAppAbierta = "Ninguna";

}

public int CanalActual

{

get => CanalActual;

set

{

```
if (value >= 1 & value <= 100)  
    canalActual = value;  
else  
    console.WriteLine("Canal fuera de Rango.");  
}
```

{

}

Public int Volumen

{

get => Volumen;
set

{

if (value >= 0 & value <= 100)

volumen = value;

else

console.WriteLine("Volumen debe estar 0, 100");

}

}

Public void Encender ()

{

encendido = true;

console.WriteLine("televisor encendido.");

}

Public void Apagar ()

{

encendido = false;

console.WriteLine("apagado");

}

Public void CambiarCanal (int Canal)

{

if (!encendido)

{

console.WriteLine("encide el televisor.");

return;

}

CanalActual = Canal;

Console.WriteLine("Canal cambia a {actual}");

}

Public void SubirVolumen ()

{

IF (modoSilencio)

{

o Veterinaria

```
Using System;
Using System.Text.Json; (2)

name space Mixta

public class Veterinaria
{
    public string nombre { get; set; }
    public string direccion { get; set; }
    public string telefono { get; set; }
    public int CapacidadAnimales { get; set; }
    public bool Abierta { get; set; }

    private double ingresosMensuales;
    private int pacientesAtendidos;
    private JsonDocument registroGeneral;
    private byte fotoClinica;
    private bool sistemaOnline;

    protected string VeterinariaPrincipal;
    protected int consultoriosActivos;
    protected bool EmergenciasActivas;
    protected double precioConsulta;
    protected string tipoVeterinaria;

    public Veterinaria(string nombre, string direccion, string telefono,
        int capacidad, bool abierta)
    {
        Nombre = nombre;
        Direccion = direccion;
        Telefono = telefono;
        CapacidadAnimales = Capacidad;
        Abierta = false;

        IngresosMensuales = 0;
        PacientesAtendidos = 0;
        RegistroGeneral = JsonDocument.Parse("{}");
        FotoClinica = array.Empty<byte>();
        SistemaOnline = true;

        VeterinariaPrincipal = "sin asignar";
        ConsultoriosActivos = 2;
        EmergenciasActivas = false;
        PrecioConsulta = precioConsulta;
        TipoVeterinaria = "General";
    }

    public double GetIngresos() => IngresosMensuales;

    public void SetIngresos(double valor)
    {
        if (valor < 0)
            Console.WriteLine("Ingresos no puede ser Negativos.");
    }
}
```

```

class Veterinaria {
    int IngresosMensuales = 0;
    bool Abierta = true;

    public void Abrir() {
        if (!Abierta) {
            Abierta = true;
            Console.WriteLine("Veterinaria abierta.");
        }
    }

    else {
        Console.WriteLine("abierta");
    }

    public void RegistrarConsulta(string nombreMascota) {
        if (Abierta) {
            pacientesAtendidos++;
            IngresosMensuales += precioConsulta;
            Console.WriteLine($"Consulta registrada para {nombreMascota}. Total {Mascota}: {MascotaAtendidas}");
        }
    }

    else {
        Console.WriteLine("Veterinaria Cerrada");
    }

    public void ActivarEmergencia() {
        EmergenciaActivas = true;
        Console.WriteLine("Emergencia Activada.");
    }

    public void Cerrar() {
        Abierta = false;
        Console.WriteLine("Veterinaria ha cerrado.");
    }

    public void Cerrar() {
        Abierta = false;
    }
}

```

farmacia

```
Using System;
Using System.Text.Json;
namespace Mixto
```

(3)

{

```
public class farmacia
```

{

```
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 VentaRealizada;
```

```
private bool SistemaActivo;
```

```
private byte LogoFarmacia;
```

```
private JSONDocument registroVentas;
```

```
protected double PrecioPromedio;
```

```
protected int ProductoDisponibles;
```

```
protected bool TurnoNoche;
```

```
protected string tipoFarmacia;
```

```
protected string proveedorPrincipal;
```

```
public farmacia (string nombre, string direccion, string encargado)
```

{

```
Nombre = nombre;
```

```
Direccion = direccion;
```

```
Encargado = encargado;
```

```
Abierta = false;
```

```
NumeroEmpleados = 0;
```

```
IngresosDiarios = 0;
```

```
VentasRealizadas = 0;
```

```
SistemaActivo = true;
```

```
LogoFarmacia = Array.Empty<byte>();
```

```
registroVentas = JSONDocument.Parse ("{}");
```

```
PrecioPromedio = 25000;
```

```
ProductoDisponible = 150;
```

```
TurnoNombre = false;
```

```
TipoFarmacia = "popular";
```

```
proveedorPrincipal = "distribuidora";
```

}

```
public double GetIngresos () => IngresosDiarios;
```

```
public void SetIngresos (double valor)
```

{

```
if (valor >= 0)
```

```
IngresosDiarios = valor;
```

```
else
```

```
Console.WriteLine ("Valor no puede ser Negativo.");
```

}

```

public void Abrir()
{
    Abierta = true;
    Console.WriteLine("Farmacia Abierta.");
}

public void RegistrarVenta(string producto, double precio)
{
    public void RegistrarVenta(string productos, double precio)
    {
        if (Abierta & productoDisponibles > 0)
        {
            VentaRealizadas++;
            IngresosDiarios += precio;
            productoDisponibles--;
            Console.WriteLine($"Venta registrada: {producto} ({${precio}})");
        }
        else
        {
            Console.WriteLine("No se puede registrar la Venta");
        }
    }
}

public void Cerrar()
{
    Abierta = false;
    Console.WriteLine("Farmacia Cerrada.");
}

```

```

modoSilencio = false;
Console.WriteLine("modo Silencio desactivado");

}

volumen +=;
Console.WriteLine("Volumen: { Volumen }");

}

public void Silenciar()
{
    modoSilencio = true;
    Console.WriteLine("televisor modo silencio");
}

}

public void ConectarWifi()
{
    ConexionWifi = true;
    Console.WriteLine("televisor Conectado a Wifi");
}

}

public void AbrirApp(string nombreApp)
{
    if (!encendido)
    {
        Console.WriteLine("encende el televisor");
        return;
    }

    UltimaAppAbierta = nombreApp;
    Console.WriteLine("abriendo App { nombreApp }");
}

}

● Restaurante (4)

using System;
using System.Text.Json;
namespace Mixto

{

public class Restaurante
{
    public string Nombre { get; set; }
    public string Direccion { get; set; }
    public string tipoComida { get; set; }
    public bool Abierto { get; set; }
    public int NumeroMasas { get; set; }

    private double IngresosDia;
    private int pedidosCompletados;
    private cocinadictiva;
    private byte logoRestaurante;
}

```

private JSON Document ManoJSON;

protected String ChefPrincipal;
protected int Empleados;
protected double Calificacion;
protected bool ServicioDomicilio;
protected String ProveedorAlimentos;

public Restaurante (String nombre, String direccion, String tipoComida)

{

Nombre = nombre;
Direccion = direccion;
tipoComida = tipoComida;
Abierto = false;
NumeroMesa = 10;

IngresosDia = 0;
pedidosCompletados = 0;

CocinaActivo = false;

LogoRestaurante = array [empty <byte>[]];

menuJSON = JSON Document.parse ("{" Mano "}; {" }");

ChefPrincipal = "Carlos Ramirez";

Empleados = 5;

Calificacion = 4.5;

ServicioDomicilio = true;

ProveedorAlimentos = Distribuidora Gourmet S.A.;

}

public double GetIngresosDia () => IngresosDia;

public void SetIngresosDia (double valor)

{

If (valor >= 0)

IngresosDia = valor;

else

Console.WriteLine ("Ingresos NO Pueden Ser Negativos");

}

public void Abrir ()

{

Abrirta = true;

CocinaActiva = true;

Console.WriteLine ("Abierto");

}

public void RegistrarPedido (String Plato, double precio)

{

If (Abierto & & CocinaActiva)

pedidosCompletados += 1;

IngresosDia += precio;

Console.WriteLine ("El pedido " + Plato + " completo " + (\$ & precio) + ");

}

public void Correr ()

Abierto = false;
Cocina Activa = false;
Console.WriteLine ("Restaurante ha cerrado.");

• tienda de Ropa (5)

```
using System;
using System.Text.Json;
namespace Mixlo
```

```
public class tiendaRopa
```

```
public string Nombre { get; set; }
public string Direccion { get; set; }
public string Encargado { get; set; }
public bool Abierto { get; set; }
public int Empleados { get; set; }

private double IngresosDiarios;
private int ProductosVendidos;
private bool SistemaActivo;
private byte LogoTienda;
private JsonDocument catalogoJSON;

protected String proveedorPrincipal;
protected double precioPromedio;
protected int StockDisponible;
protected bool promocionActiva;
protected string tipoTienda;
```

```
public tiendaRopa (string nombre, string direccion, string encargado)
```

Nombre = nombre;
Direccion = direccion;
Encargado = encargado;
Abierto = false;
Empleados = 4;

IngresosDiarios = 0;
ProductosVendidos = 0;
SistemaActivo = true;
LogoTienda = Array.Empty<byte>();
catalogoJSON = JsonDocument.Parse ("{"Catalogo": []}");

proveedorPrincipal = "textiles driven";
precioPromedio = 8500;
StockDisponible = 300;
promocionActiva = true;
tipoTienda = "Moda Casual";

3

```
public double GetIngresos () => IngresosDiarios;
```

```
public void SetIngresos (double valor)
```

```
{
```

```
if (valor >= 0)
```

```
ingresosDiarios = valor
```

```
else
```

```
controlle.WriteLine ("valor no debe ser Negativo.");
```

```
}
```

```
public void Abrir ()
```

```
{
```

```
Abierta = true;
```

```
console.WriteLine ("Abierto");
```

```
}
```

```
public void RegistrarVentas (String prendo, double precio)
```

```
{
```

```
if (Abierta & SistemaActivo & stockDisponible > 0)
```

```
{
```

```
prendasVendidas ++;
```

```
IngresosDiarios += precio;
```

```
stockDisponible --;
```

```
console.WriteLine ("Venta Realizada { prendo } ($ { precio })");
```

```
}
```

```
else
```

```
{
```

```
console.WriteLine ("No se puede vender ({ cerrar o sin stock })");
```

```
}
```

```
}
```

```
public void Cerrar ()
```

```
{
```

```
Abierta = false;
```

```
console.WriteLine ("Cerrado");
```

```
}
```

```
}
```

```
}
```

•Carro Inteligente

Using System;

Using System.Text.Json

namespace Mixto

public class CarroInteligente

```
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 UsandoJsonConfiguracion;
```

```
private local gpsActivo;
```

```
private byte registroAudio;
```

```
protected bool CinturonesAbrochados;
```

```
protected bool LucesEncendidas;
```

```
protected string modoConduccion;
```

```
protected int PuertasCerradas;
```

```
protected double Kilometraje;
```

```
public CarroInteligente (string marca, modelo, string color, double  
VelocidadMaxima);
```

```
{
```

```
marca = marca;
```

```
modelo = modelo;
```

```
color = color;
```

```
VelocidadMaxima = maxima = Velocidadmaxima;
```

```
encendido = false;
```

```
velocidadActual = 0;
```

```
Combustible = 100;
```

```
gpsActivo = false;
```

```
registroAudio = array.Empty < byte > (1);
```

```
Configuracion = UsandoJsonConfiguracion.Parse ("");
```

```
modoConduccion = "Normal";
```

```
Kilometraje = 0;
```

```
public double GetCombustible () => Combustible;
```

```
public void SetCombustible (double Valor)
```

```
{
```

```
if (Valor < 0 || Valor > 100)
```

```
Console.WriteLine ("Inval de combustible Invalido!");
```

```
else
```

```
combustible = Valor;
```

```
}
```

```

public void encender ()
{
    if (!encendido)
    {
        encendido = true;
        console.WriteLine ("Carro encendido");
    }
    else
        console.WriteLine ("estaba encendido.");
}

public void Acelerar (double Velocidad)
{
    if (encendido & Combustible > 0)
    {
        velocidadActual = Math.Min (velocidad, VelocidadMaxima);
        combustible -= 5;
        Kilometraje += velocidadActual * 0.1;
        console.WriteLine ("");
    }
    else
        console.WriteLine ("No puedes acelerar. encendido. combustible.");
}

public void frenar ()
{
    if (velocidadActual > 0)
        console.WriteLine ("esta detenido.");
}

public void ActivarLuces ()
{
    lucesEncendidas = LucesEncendidas + 1;
    console.WriteLine ("luces {" + lucesEncendidas + "}");
}

```