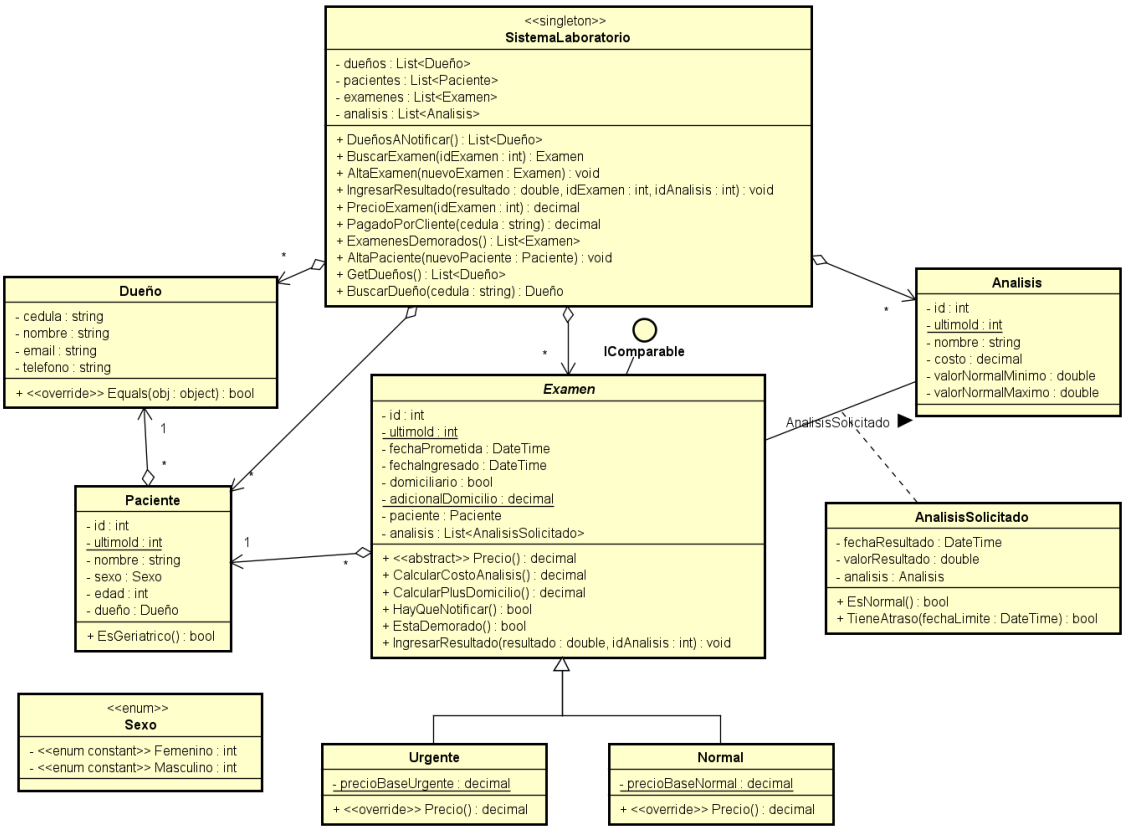

Parte 2 a.:

Listado de dueños de los pacientes a los que es necesario notificar por tener exámenes con resultados anormales. (**15 puntos**)

CLASE SistemaLaboratorio

```
    public List<Dueño> DueñosANotificar()
    {
        List<Dueño> retorno = new List<Dueño>();
        foreach (Examen unExamen in _examenes)
        {
            if (unExamen.HayQueNotificar() &&
!retorno.Contains(unExamen.Paciente.Dueño))
            {
                retorno.Add(unExamen.Paciente.Dueño);
            }
        }
        return retorno;
    }
```

CLASE Examen

```
    public bool HayQueNotificar()
    {
        int i = 0; bool anormal = false;

        while (i < _analisis.Count && !anormal)
        {
            if (!_analisis[i].EsNormal())
            {
                anormal = true;
            }
            i++;
```

```
        }
        return anormal;
    }
```

CLASE AnalisisSolicitado

```
    public bool EsNormal()
    {
        return _valorResultado >= _analisis.ValorMinimo && _valorResultado
<= _analisis.ValorMaximo;
    }
```

CLASE Dueño

```
    public override bool Equals(object obj)
    {
        return obj is Dueño otroD && _cedula.Equals(otroD._cedula);
    }
```

Parte 2 b.:

Dado un número de examen, calcular su precio total a pagar. (**20 puntos**)

CLASE SistemaLaboratorio

```
    public decimal PrecioExamen(int idExamen)
    {
        return BuscarExamen(idExamen).Precio();
    }



    public Examen BuscarExamen(int idExamen)
    {
        int i = 0;
        Examen buscado = null;
        while (i <= _examenes.Count && buscado != null)
        {
            if (_examenes[i].Id == idExamen)
            {
                buscado = _examenes[i];
            }
            i++;
        }
        if (buscado == null)
            throw new Exception("El examen buscado no existe");

        return buscado;
    }
```

CLASE Examen

```
    public abstract decimal Precio();

    public decimal CalcularCostoAnalisis()
    {
        decimal precio = 0;
        foreach (AnalisisSolicitado unAnalisis in _analisis)
        {
            precio += unAnalisis.Analisis.Costo;
        }
        return precio;
```

```
}

public decimal CalcularPlusDomicilio()
{
    decimal adicionalDomicilio = 0;
    if (_domiciliario)
    {
        adicionalDomicilio += s_adicionalDomicilio;
    }
    return adicionalDomicilio;
}
```

CLASE Normal : Examen

```
public override decimal Precio()
{
    decimal precio = CalcularCostoAnalisis();
    precio += s_precioBaseNormal;
    if (this.Paciente.EsGeriatrico())
    {
        precio -= precio * 0.05;
    }
    return precio + CalcularPlusDomicilio();
}
```

CLASE Urgente : Examen

```
public override decimal Precio()
{
    decimal precio = CalcularCostoAnalisis();
    precio += s_precioBaseUrgente;
    precio += precio * 0.05;
    return precio + CalcularPlusDomicilio();
}
```

Parte 2 c.:

Obtener los exámenes que presentan demoras en la entrega de resultados. Se ordenarán descendentemente por fecha prevista de resultado. (**15 puntos**)

CLASE SistemaLaboratorio

```
public List<Examen> ExamenesDemorados()
{
    List<Examen> retorno = new List<Examen>();
    foreach (Examen unExamen in _examenes)
    {
        if (unExamen.EstaDemorado())
        {
            retorno.Add(unExamen);
        }
    }
    retorno.Sort();
    return retorno;
}
```

CLASE Examen : IComparable<Examen>

```
public bool EstaDemorado()
{
    bool demorado = false;
    if (DateTime.Today <= _fechaPrometida)
    {
        int i = 0;
```

```
            while (i < _analisis.Count && !demorado)
            {
                if (_analisis[i].TieneAtraso(_fechaPrometida))
                    demorado = true;
                i++;
            }
        }
        return demorado;
    }

    public int CompareTo(Examen? otroE)
    {
        if (otroE == null)
            return 1;
        return otroE._fechaPrometida.Date.CompareTo(_fechaPrometida.Date)
* -1;
    }
```

CLASE AnalisisSolicitado

```
    public bool TieneAtraso(DateTime fechaLimite)
    {
        return _fechaResultado.Date > fechaLimite.Date;
    }
```

Parte 3 a.:

Agregar un nuevo paciente al sistema (un animal doméstico). No se deberá implementar el código de dominio necesario para resolver el requerimiento, pero sí se incluirá en el diagrama de clases la firma del método (o métodos). (**15 puntos**)

CONTROLADOR Pacientes

```
    private SistemaLaboratorio _unS = SistemaLaboratorio.Instancia;

    public IActionResult Alta()
    {
        ViewBag.Dueños = _unS.GetDueños();
        return View(new Paciente());
    }

    [HttpPost]
    public IActionResult Alta(Paciente nuevoPaciente)
    {
        try
        {
            nuevoPaciente.Dueño =
_unS.BuscarDueño(nuevoPaciente.Dueño.Cedula);
            _unS.AltaPaciente(nuevoPaciente);
        }
        catch (Exception e)
        {
            ViewBag.Dueños = _unS.GetDueños();
            ViewBag.Mensaje = e.Message;
            return View(nuevoPaciente);
        }
        ViewBag.Dueños = _unS.GetDueños();
        return RedirectToAction("Index", new { mensaje = "Alta exitosa"
});
    }
```

VISTA Alta

```
@model Paciente
```

```
<h1>Alta de paciente</h1>

@if (ViewBag.Mensaje != null)
{
    <div>
        <p>@ViewBag.Mensaje</p>
    </div>
}

<form method="post">
    <label>Nombre:</label>
    <input type="text" name="Nombre" value="@Model.Nombre"/>
    <label>Sexo:</label>
    <select name="Sexo">
        <option value="0">Femenino</option>
        <option value="1">Masculino</option>
    </select>
    <label>Edad:</label>
    <input type="number" name="Edad" value="@Model.Edad" />
    <label>Dueño:</label>
    <select name="Dueño">
        @foreach (Dueño unDueño in ViewBag.Dueños)
        {
            <option value="@unDueño.Cedula">@unDueño.Nombre</option>
        }
    </select>
    <input type="submit" name="Ingresar" />
</form>
```

Parte 3 b.:

Dado el id de un examen, listar todos los análisis que incluye. Se mostrarán el id y el nombre del análisis. (**10 puntos**)

CONTROLADOR Examenes

```
    public IActionResult Detalle(int id)
    {
        Examen examen = null;
        try
        {
            Examen = _unS.BuscarExamen(id);
        }catch (Exception e)
        {
            ViewBag.Mensaje = e.Message;
        }
        return View(examen);
    }
```

VISTA Detalle

```
@model Examen

<h1>Detalle de examen</h1>

@if (ViewBag.Mensaje != null)
{
    <div>
        <p>@ViewBag.Mensaje</p>
    </div>
}
```

```
else
{
    <h2>Análisis del examen @Model.Id</h2>
    <table>
        <thead>
            <tr>
                <th>Id</th>
                <th>Nombre</th>
            </tr>
        </thead>
        <tbody>
            @foreach (AnalisisSolicitado analisisSolicitado in
Model.Analisis)
            {
                <tr>
                    <td>@analisisSolicitado.Analisis.Id</td>
                    <td>@analisisSolicitado.Analisis.Nombre</td>
                </tr>
            }
        </tbody>
    </table>
}
```