Universidad Tecnológica Nacional **Facultad Regional Avellaneda**	
Técnico Superior en Programación - Técnico Superior en Sistemas Informáticos	
Materia: LABORATORIO DE PROGRAMACIÓN II	
Instancia[(1)]:	**TP Nº3**

Generar una Solución nombrada como: *Apellido.Nombre.Division.TP3* con las siguientes características:

Condiciones de corrección y aprobación:

1. Qué se respeten todas las consignas dadas.
2. Qué todas las Clases, Métodos, Atributos, Propiedades, etc. sean nombrados exactamente como fue pedido en el enunciado.
3. Qué se introduzca el código de la función Main dada sin modificaciones.
4. Qué el proyecto no contenga errores de ningún tipo.
5. Qué el código compile y se ejecute de manera correcta.
6. Qué la salida por pantalla con el formato de la entregada en este mismo documento.
7. Se deberá reutilizar código cada vez que se pueda, aunque no esté explicitado en el contenido del texto.
8. Se deberá documentar el código según las reglas de estilo de la cátedra.
9. Generar al menos dos test unitario que validen Excepciones, uno que valide un valor numérico y uno que valide que no haya valores nulos en algún atributo de las clases dadas.
10. Qué pase, sin modificaciones de código, los Test Unitarios que genere quien esté a cargo de la corrección del examen (para eso se deberá cumplir con todo lo anteriormente planteado).

Características:

Clase Persona:

- Abstracta, con los atributos Nombre, Apellido, Nacionalidad y DNI.
- Se deberá validar que el DNI sea correcto, teniendo en cuenta su nacionalidad. Argentino entre 1 y 89999999. Caso contrario, se lanzará la excepción DniInvalidoException.
- Sólo se realizarán las validaciones dentro de las propiedades.
- Validará que los nombres sean cadenas con caracteres válidos para nombres. Caso contrario, no se cargará.
- ToString retornará los datos de la Persona.

Clase PersonaGimnasio:

- Abstracta, con el atributo Id.
- Método protegido y virtual MostrarDatos retornará todos los datos de la PersonaGimnasio.
- Método protegido y abstracto ParticiparEnClase.
- Dos PersonaGimnasio serán iguales si y sólo si son del mismo Tipo y su Id o DNI son iguales.

Clase Alumno:

- Atributos ClaseQueToma del tipo EClase y EstadoCuenta del tipo EEstadoCuenta.
- Sobreescribirá el método MostrarDatos con todos los datos del alumno.
- ParticiparEnClase retornará la cadena "TOMA CLASE DE " junto al nombre de la clase que toma.
- ToString hará públicos los datos del Alumno.
- Un Alumno será igual a un EClase si toma esa clase y su estado de cuenta no es Deudor.
- Un Alumno será disntito a un EClase sólo si no toma esa clase.

Clase Instructor:

- Atributos ClasesDelDia del tipo Cola y random del tipo Random y estático.
- Sobreescribirá el método MostrarDatos con todos los datos del alumno.
- ParticiparEnClase retornará la cadena "CLASES DEL DÍA " junto al nombre de la clases que da.
- ToString hará públicos los datos del Instructor.
- Se inicializará a random sólo en un constructor.
- En el constructor de instancia se inicializará ClasesDelDia y se asignarán dos clases al azar al instructor mediante el método _randomClases. Las dos clases pueden o no ser la misma.
- Un Instructor será igual a un EClase si da esa clase.

Clase Jornada:

- Atributos Instructor, Clase y Alumnos que toman dicha clase.
- Se inicializará la lista de alumnos en el constructor por defecto.
- Una Jornada será igual a un Alumno si el mismo participa de la clase.
- Agregar Alumnos a la clase por medio del operador +, validando que no estén previamente cargados.
- ToString mostrará todos los datos de la Jornada.
- Guardar de clase guardará los datos de la Jornada en un archivo de texto.
- Leer de clase retornará los datos de la Jornada como texto.

Clase Gimnasio:

- Atributos Alumnos (lista de inscriptos), Instructores (lista de quienes pueden dar clases) y Jornadas.
- Se accederá a una Jornada específica a través de un indexador.
- Un Gimnasio será igual a un Alumno si el mismo está inscripto en él.
- Un Gimnasio será igual a un Instructor si el mismo está dando clases en él.
- Al agregar una clase a un Gimnasio se deberá generar y agregar una nueva Jornada indicando la clase, un Instructor que pueda darla (según su atributo ClasesDelDia) y la lista de alumnos que la toman (todos los que coincidan en su campo ClaseQueToma).
- Se agregarán Alumnos e Instructores mediante el operador +, validando que no estén previamente cargados.
- La igualación entre un Gimnasio y una Clase retornará el primer instructor capaz de dar esa clase. Sino, lanzará la Excepción SinInstructorException. El distinto retornará el primer Instructor que no pueda dar la clase.
- MostrarDatos será privado y de clase. Los datos del Gimnasio se harán públicos mediante ToString.

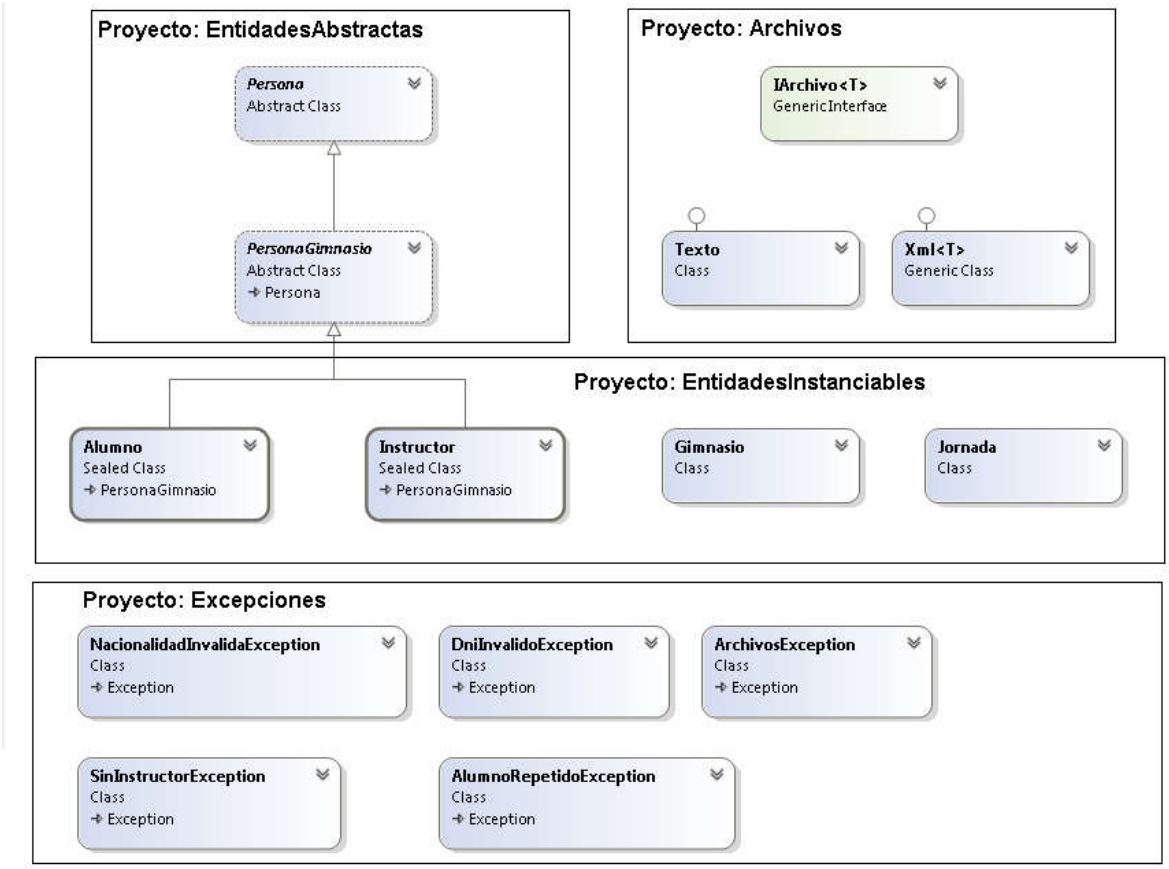
- Guardar de clase serializará los datos del Gimnasio en un XML, incluyendo todos los datos de sus Instructores, Alumnos y Jornadas.
- Leer de clase retornará un Gimnasio con todos los datos previamente serializados.

Archivos:

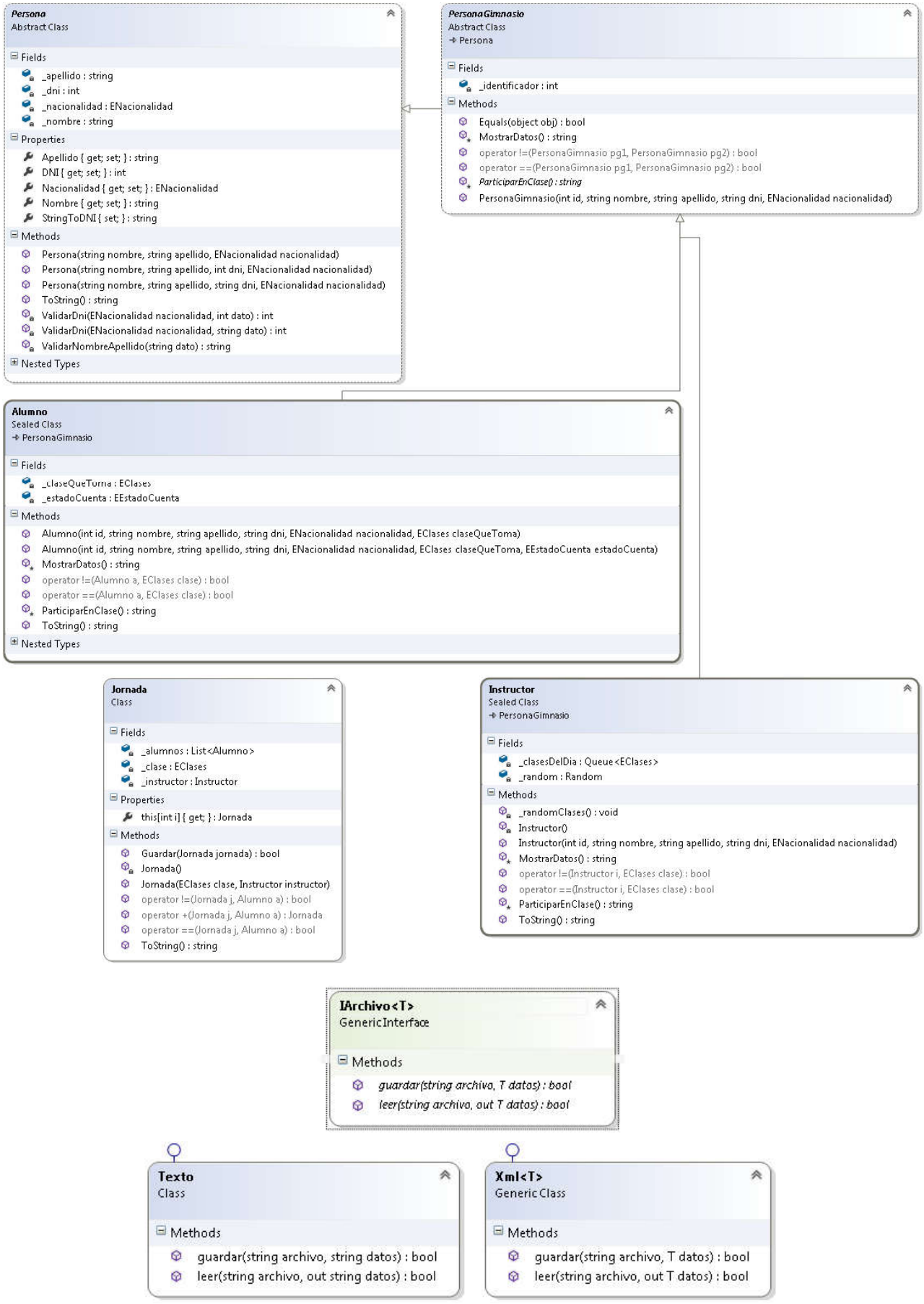
- Generar una interfaz con las firmas para guardar y leer.
- Implementar la interfaz en las clases Xml y Texto, a fin de poder guardar y leer archivos de esos tipos.

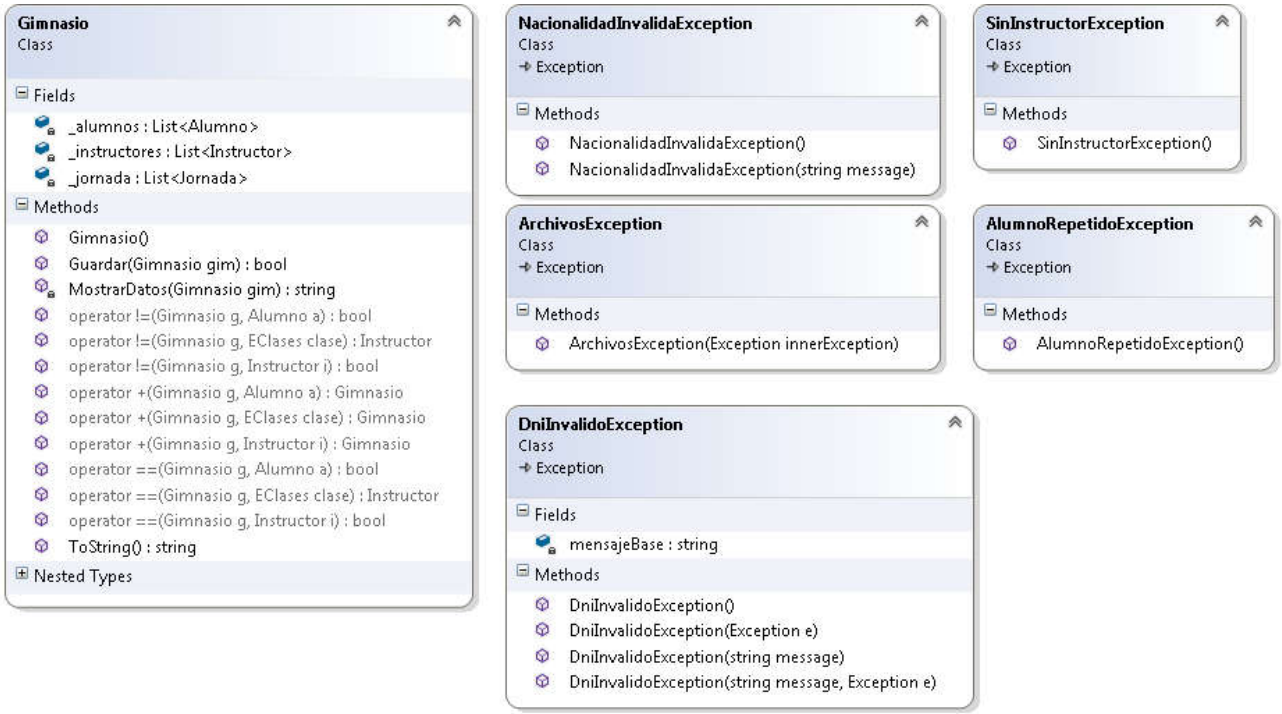
Diagrama de clases:

Se detallan los nombres de las clases, divididas por el Proyecto y Namespace que las contendrá.



Detalle de las clases:




Gimnasio
Class
Fields
_alumnos : List<Alumno>
_instructores : List<Instructor>
_jornada : List<Jornada>
Methods
Gimnasio()
Guardar(Gimnasio gim) : bool
MostrarDatos(Gimnasio gim) : string
operator !=(Gimnasio g, Alumno a) : bool
operator !=(Gimnasio g, EClases clase) : Instructor
operator !=(Gimnasio g, Instructor i) : bool
operator +(Gimnasio g, Alumno a) : Gimnasio
operator +(Gimnasio g, EClases clase) : Gimnasio
operator +(Gimnasio g, Instructor i) : Gimnasio
operator ==(Gimnasio g, Alumno a) : bool
operator ==(Gimnasio g, EClases clase) : Instructor
operator ==(Gimnasio g, Instructor i) : bool
ToString() : string
Nested Types
NacionalidadInvalidaException
Class
Exception
Methods
NacionalidadInvalidaException()
NacionalidadInvalidaException(string message)
SinInstructorException
Class
Exception
Methods
SinInstructorException()
ArchivosException
Class
Exception
Methods
ArchivosException(Exception innerException)
AlumnoRepetidoException
Class
Exception
Methods
AlumnoRepetidoException()
DniInvalidoException
Class
Exception
Fields
mensajeBase : string
Methods
DniInvalidoException()
DniInvalidoException(Exception e)
DniInvalidoException(string message)
DniInvalidoException(string message, Exception e)

Salida por pantalla:

```
La nacionalidad no se condice con el número de DNI
Alumno repetido.
No hay instructor para la clase.
No hay instructor para la clase.
JORNADA:
CLASE DE Natacion POR NOMBRE COMPLETO: Juarez, Roberto
NACIONALIDAD: Argentino

CARNET NÚMERO: 2
CLASES DEL DÍA:
Natacion
Pilates

ALUMNOS:
NOMBRE COMPLETO: Suarez, Joaquin
NACIONALIDAD: Extranjero

CARNET NÚMERO: 7

ESTADO DE CUENTA: Cuota al día
TOMA CLASES DE Natacion
<------------------------------------------------>

CLASE DE Pilates POR NOMBRE COMPLETO: Lopez, Juan
NACIONALIDAD: Argentino

CARNET NÚMERO: 1
CLASES DEL DÍA:
Pilates
Pilates

ALUMNOS:
NOMBRE COMPLETO: Hernandez, Miguel
NACIONALIDAD: Extranjero

CARNET NÚMERO: 4

ESTADO DE CUENTA: Cuota al día
TOMA CLASES DE Pilates
NOMBRE COMPLETO: Smith, Rodrigo
NACIONALIDAD: Argentino

CARNET NÚMERO: 8

ESTADO DE CUENTA: Cuota al día
TOMA CLASES DE Pilates
<------------------------------------------------>


Archivo de Gimnasio guardado
Archivo de Jornada 0 guardado
```

Main

```
static void Main(string[] args)
{
    Gimnasio gim = new Gimnasio();

    Alumno a1 = new Alumno(1, "Juan", "Lopez", "12234456",
EntidadesAbstractas.Persona.ENacionalidad.Argentino, Gimnasio.EClases.CrossFit,
Alumno.EEstadoCuenta.MesPrueba);
    gim += a1;
    try
    {
        Alumno a2 = new Alumno(2, "Juana", "Martinez", "12234458",
EntidadesAbstractas.Persona.ENacionalidad.Extranjero, Gimnasio.EClases.Natacion,
Alumno.EEstadoCuenta.Deudor);
        gim += a2;
    }
    catch (NacionalidadInvalidaException e)
    {
        Console.WriteLine(e.Message);
    }
    try
    {
        Alumno a3 = new Alumno(3, "José", "Gutierrez", "12234456",
EntidadesAbstractas.Persona.ENacionalidad.Argentino, Gimnasio.EClases.CrossFit,
Alumno.EEstadoCuenta.MesPrueba);
        gim += a3;
    }
    catch (AlumnoRepetidoException e)
    {
        Console.WriteLine(e.Message);
    }
    Alumno a4 = new Alumno(4, "Miguel", "Hernandez", "92264456",
EntidadesAbstractas.Persona.ENacionalidad.Extranjero, Gimnasio.EClases.Pilates,
Alumno.EEstadoCuenta.AlDia);
    gim += a4;
    Alumno a5 = new Alumno(5, "Carlos", "Gonzalez", "12236456",
EntidadesAbstractas.Persona.ENacionalidad.Argentino, Gimnasio.EClases.CrossFit,
Alumno.EEstadoCuenta.AlDia);
    gim += a5;
    Alumno a6 = new Alumno(6, "Juan", "Perez", "12234656",
EntidadesAbstractas.Persona.ENacionalidad.Argentino, Gimnasio.EClases.Natacion,
Alumno.EEstadoCuenta.Deudor);
    gim += a6;
    Alumno a7 = new Alumno(7, "Joaquin", "Suarez", "91122456",
EntidadesAbstractas.Persona.ENacionalidad.Extranjero, Gimnasio.EClases.Natacion,
Alumno.EEstadoCuenta.AlDia);
    gim += a7;
    Alumno a8 = new Alumno(8, "Rodrigo", "Smith", "22236456",
EntidadesAbstractas.Persona.ENacionalidad.Argentino, Gimnasio.EClases.Pilates,
Alumno.EEstadoCuenta.AlDia);
    gim += a8;

    Instructor i1 = new Instructor(1, "Juan", "Lopez", "12234456",
EntidadesAbstractas.Persona.ENacionalidad.Argentino);
    gim += i1;
    Instructor i2 = new Instructor(2, "Roberto", "Juarez", "32234456",
EntidadesAbstractas.Persona.ENacionalidad.Argentino);
    gim += i2;

    try
    {
        gim += Gimnasio.EClases.CrossFit;
    }
    catch (SinInstructorException e)
    {
        Console.WriteLine(e.Message);
    }
```

```
        try
        {
            gim += Gimnasio.EClases.Natacion;
        }
        catch (SinInstructorException e)
        {
            Console.WriteLine(e.Message);
        }
        try
        {
            gim += Gimnasio.EClases.Pilates;
        }
        catch (SinInstructorException e)
        {
            Console.WriteLine(e.Message);
        }
        try
        {
            gim += Gimnasio.EClases.Yoga;
        }
        catch (SinInstructorException e)
        {
            Console.WriteLine(e.Message);
        }

        Console.WriteLine(gim.ToString());

        Console.ReadKey();

        try
        {
            Gimnasio.Guardar(gim);
            Console.WriteLine("Archivo de Gimnasio guardado");
        }
        catch (ArchivosException e)
        {
            Console.WriteLine(e.Message);
        }
        try
        {
            int jornada = 0;
            Jornada.Guardar(gim[jornada]);
            Console.WriteLine("Archivo de Jornada {0} guardado", jornada);
        }
        catch (ArchivosException e)
        {
            Console.WriteLine(e.Message);
        }

        Console.ReadKey();
    }
```