


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 **NO** modifique la función Main dada.
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. Test Unitarios:
 - a. Generar al menos dos test unitario que validen Excepciones
 - b. Generar al menos uno que valide un valor numérico
 - c. Generar al menos uno que valide que no haya valores nulos en algún atributo de las clases dadas.
10. Todas las excepciones deberán tener mensajes propios.
11. 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 Universitario:

- Abstracta, con el atributo Legajo.
- Método protegido y virtual MostrarDatos retornará todos los datos del Universitario.
- Método protegido y abstracto ParticiparEnClase.
- Dos Universitario serán iguales si y sólo si son del mismo Tipo y su Legajo 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 distinto a un EClase sólo si no toma esa clase.

Clase Profesor:

- Atributos ClasesDelDia del tipo Cola y random del tipo Random y estático.
- Sobrescribir 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 Profesor.
- Se inicializará a Random sólo en un constructor.
- En el constructor de instancia se inicializará ClasesDelDia y se asignarán dos clases al azar al Profesor mediante el método randomClases. Las dos clases pueden o no ser la misma.
- Un Profesor será igual a un EClase si da esa clase.

Clase Jornada:

- Atributos Profesor, 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 Universidad:

- Atributos Alumnos (lista de inscriptos), Profesores (lista de quienes pueden dar clases) y Jornadas.
- Se accederá a una Jornada específica a través de un indexador.
- Un Universidad será igual a un Alumno si el mismo está inscripto en él.
- Un Universidad será igual a un Profesor si el mismo está dando clases en él.
- Al agregar una clase a un Universidad se deberá generar y agregar una nueva Jornada indicando la clase, un Profesor 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 y Profesores mediante el operador +, validando que no estén previamente cargados.

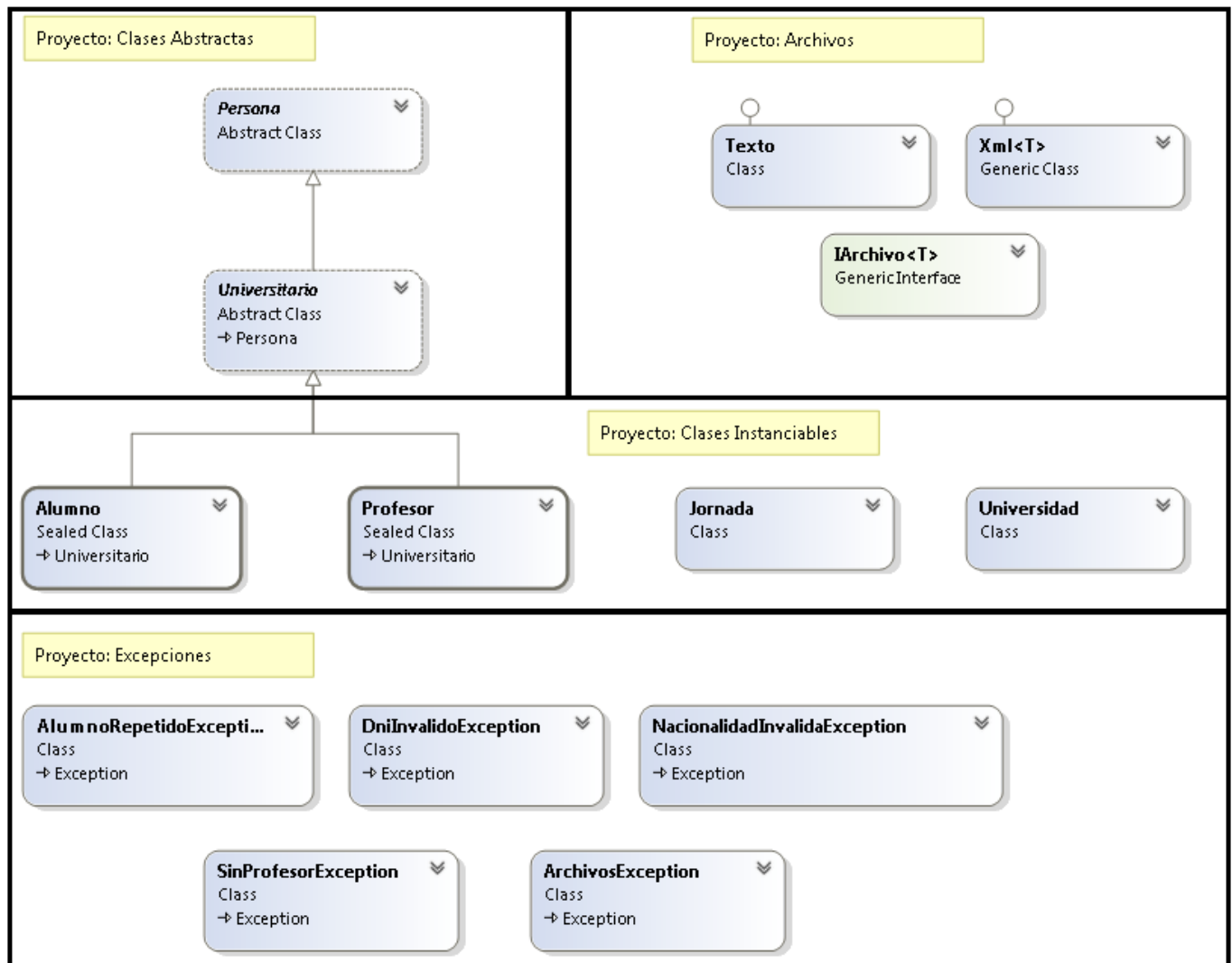
- La igualdad entre un Universidad y una Clase retornará el primer Profesor capaz de dar esa clase. Sino, lanzará la Excepción SinProfesorException. El distinto retornará el primer Profesor que no pueda dar la clase.
- MostrarDatos será privado y de clase. Los datos del Universidad se harán públicos mediante ToString.
- Guardar de clase serializará los datos del Universidad en un XML, incluyendo todos los datos de sus Profesores, Alumnos y Jornadas.
- Leer de clase retornará un Universidad 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:

ArchivosException
Class
→ Exception

Methods

- ArchivosException(Exception innerException)

NacionalidadInvalidaException
Class
→ Exception

Methods

- NacionalidadInvalidaException()
- NacionalidadInvalidaException(string message)

DniInvalidoException
Class
→ Exception

Fields

- mensajeBase : string

Methods

- DniInvalidoException()
- DniInvalidoException(Exception e)
- DniInvalidoException(string message)
- DniInvalidoException(string message, Exception e)

AlumnoRepetidoException
Class
→ Exception

Methods

- AlumnoRepetidoException()

SinProfesorException
Class
→ Exception

Methods

- SinProfesorException()

Texto
Class

Methods

- guardar(string archivo, string datos) : bool
- leer(string archivo, out string datos) : bool

Xml<T>
Generic Class

Methods

- guardar(string archivo, T datos) : bool
- leer(string archivo, out T datos) : bool

IArchivo<T>
GenericInterface

Methods

- guardar(string archivo, T datos) : bool*
- leer(string archivo, out T datos) : bool*

Persona

Abstract Class

Fields

- `_apellido : string`
- `_dni : int`
- `_nacionalidad : ENacionalidad`
- `_nombre : string`

Properties

- `Apellido { get; set; } : string`
- `DNI { get; set; } : int`
- `Nacionalidad { get; set; } : ENacionalidad`
- `Nombre { get; set; } : string`
- `StringToDNI { set; } : string`

Methods

- `Persona()`
- `Persona(string nombre, string apellido, ENacionalidad nacionalidad)`
- `Persona(string nombre, string apellido, int dni, ENacionalidad nacionalidad)`
- `Persona(string nombre, string apellido, string dni, ENacionalidad nacionalidad)`
- `ToString() : string`
- `ValidarDni(ENacionalidad nacionalidad, int dato) : int`
- `ValidarDni(ENacionalidad nacionalidad, string dato) : int`
- `ValidarNombreApellido(string dato) : string`

Nested Types

Universitario

Abstract Class

→ Persona

Fields

- `legajo : int`

Methods

- `Equals(object obj) : bool`
- `MostrarDatos() : string`
- `operator !=(Universitario pg1, Universitario pg2) : bool`
- `operator ==(Universitario pg1, Universitario pg2) : bool`
- `ParticiparEnClase() : string`
- `Universitario()`
- `Universitario(int legajo, string nombre, string apellido, string dni, ENacionalidad nacionalidad)`

Alumno
 Sealed Class
 → Universitario

Fields

- `_claseQueToma : EClases`
- `_estadoCuenta : EEstadoCuenta`

Methods

- `Alumno()`
- `Alumno(int id, string nombre, string apellido, string dni, ENacionalidad nacionalidad, EClases claseQueToma)`
- `Alumno(int id, string nombre, string apellido, string dni, ENacionalidad nacionalidad, EClases claseQueToma, EEstadoCuenta estadoCuenta)`
- `MostrarDatos() : string`
- `operator !=(Alumno a, EClases clase) : bool`
- `operator ==(Alumno a, EClases clase) : bool`
- `ParticiparEnClase() : string`
- `ToString() : string`

Nested Types

EEstadoCuenta
 Enum

Profesor
 Sealed Class
 → Universitario

Fields

- `_clasesDelDia : Queue<EClases>`
- `_random : Random`

Methods

- `_randomClases() : void`
- `MostrarDatos() : string`
- `operator !=(Profesor i, EClases clase) : bool`
- `operator ==(Profesor i, EClases clase) : bool`
- `ParticiparEnClase() : string`
- `Profesor()`
- `Profesor()`
- `Profesor(int id, string nombre, string apellido, string dni, ENacionalidad nacionalidad)`
- `ToString() : string`

Jornada
 Class

Fields

- `_alumnos : List<Alumno>`
- `_clase : EClases`
- `_instructor : Profesor`

Properties

- `Alumnos { get; set; } : List<Alumno>`
- `Clase { get; set; } : EClases`
- `Instructor { get; set; } : Profesor`

Methods

- `Guardar(Jornada jornada) : bool`
- `Jornada()`
- `Jornada(EClases clase, Profesor instructor)`
- `Leer() : string`
- `operator !=(Jornada j, Alumno a) : bool`
- `operator +(Jornada j, Alumno a) : Jornada`
- `operator ==(Jornada j, Alumno a) : bool`
- `ToString() : string`

Universidad
 Class

Fields

- `alumnos : List<Alumno>`
- `jornada : List<Jornada>`
- `profesores : List<Profesor>`

Properties

- `Alumnos { get; set; } : List<Alumno>`
- `Instructores { get; set; } : List<Profesor>`
- `Jornadas { get; set; } : List<Jornada>`
- `this[int i] { get; set; } : Jornada`

Methods

- `Guardar(Universidad gim) : bool`
- `MostrarDatos(Universidad gim) : string`
- `operator !=(Universidad g, Alumno a) : bool`
- `operator !=(Universidad g, EClases clase) : Profesor`
- `operator !=(Universidad g, Profesor i) : bool`
- `operator +(Universidad g, Alumno a) : Universidad`
- `operator +(Universidad g, EClases clase) : Universidad`
- `operator +(Universidad g, Profesor i) : Universidad`
- `operator ==(Universidad g, Alumno a) : bool`
- `operator ==(Universidad g, EClases clase) : Profesor`
- `operator ==(Universidad g, Profesor i) : bool`
- `ToString() : string`
- `Universidad()`

Nested Types

EClases
 Enum

Salida por pantalla:

```
La nacionalidad no se condice con el número de DNI
Alumno repetido.
No hay Profesor para la clase.
No hay Profesor para la clase.
JORNADA:
CLASE DE Laboratorio POR NOMBRE COMPLETO: Lopez, Juan
NACIONALIDAD: Argentino

LEGAJO NÚMERO: 1
CLASES DEL DÍA:
Laboratorio
Laboratorio

ALUMNOS:
NOMBRE COMPLETO: Suarez, Joaquin
NACIONALIDAD: Extranjero

LEGAJO NÚMERO: 7

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

CLASE DE Legislacion POR NOMBRE COMPLETO: Juarez, Roberto
NACIONALIDAD: Argentino

LEGAJO NÚMERO: 2
CLASES DEL DÍA:
Laboratorio
Legislacion

ALUMNOS:
NOMBRE COMPLETO: Hernandez, Miguel
NACIONALIDAD: Extranjero

LEGAJO NÚMERO: 4

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

LEGAJO NÚMERO: 8

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

-
Archivo de Universidad guardado.
Archivo de Jornada 0 guardado.
-
```


Main

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

    Alumno a1 = new Alumno(1, "Juan", "Lopez", "12234456",
EntidadesAbstractas.Persona.ENacionalidad.Argentino, Universidad.EClases.Programacion,
Alumno.EEstadoCuenta.Becado);
    gim += a1;
    try
    {
        Alumno a2 = new Alumno(2, "Juana", "Martinez", "12234458",
EntidadesAbstractas.Persona.ENacionalidad.Extranjero, Universidad.EClases.Laboratorio,
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, Universidad.EClases.Programacion,
Alumno.EEstadoCuenta.Becado);
        gim += a3;
    }
    catch (AlumnoRepetidoException e)
    {
        Console.WriteLine(e.Message);
    }
    Alumno a4 = new Alumno(4, "Miguel", "Hernandez", "92264456",
EntidadesAbstractas.Persona.ENacionalidad.Extranjero, Universidad.EClases.Legislacion,
Alumno.EEstadoCuenta.Aldia);
    gim += a4;
    Alumno a5 = new Alumno(5, "Carlos", "Gonzalez", "12236456",
EntidadesAbstractas.Persona.ENacionalidad.Argentino, Universidad.EClases.Programacion,
Alumno.EEstadoCuenta.Aldia);
    gim += a5;
    Alumno a6 = new Alumno(6, "Juan", "Perez", "12234656",
EntidadesAbstractas.Persona.ENacionalidad.Argentino, Universidad.EClases.Laboratorio,
Alumno.EEstadoCuenta.Deudor);
    gim += a6;
    Alumno a7 = new Alumno(7, "Joaquin", "Suarez", "91122456",
EntidadesAbstractas.Persona.ENacionalidad.Extranjero, Universidad.EClases.Laboratorio,
Alumno.EEstadoCuenta.Aldia);
    gim += a7;
    Alumno a8 = new Alumno(8, "Rodrigo", "Smith", "22236456",
EntidadesAbstractas.Persona.ENacionalidad.Argentino, Universidad.EClases.Legislacion,
Alumno.EEstadoCuenta.Aldia);
    gim += a8;

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

    try
    {
        gim += Universidad.EClases.Programacion;
    }
    catch (SinProfesorException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

```

try
{
    gim += Universidad.EClases.Laboratorio;
}
catch (SinProfesorException e)
{
    Console.WriteLine(e.Message);
}
try
{
    gim += Universidad.EClases.Legislacion;
}
catch (SinProfesorException e)
{
    Console.WriteLine(e.Message);
}
try
{
    gim += Universidad.EClases.SPD;
}
catch (SinProfesorException e)
{
    Console.WriteLine(e.Message);
}

Console.WriteLine(gim.ToString());

Console.ReadKey();
Console.Clear();

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

Console.ReadKey();
}

```