Trabajo Práctico 4
Colecciones y Excepciones

Nota: en todos los ejercicios que requieran uso de colecciones, utilice las colecciones del namespace *System.Collections.Generic* y no las que están en *System.Collections*.

Ejercicio 1

En este ejercicio se va a simular la ocurrencia de una excepción en un punto interno de una aplicación y se verificará cómo funciona la propagación de las mismas.

Cree dos clases de excepciones *CapaAplicacionException* y *ErrorPuntualException* las cuales hereden de *ApplicationException*, que representen un error genérico de la aplicación y un error específico respectivamente.

Genere clases y objetos para crear un callstack de 5 niveles, es decir que un método de un objeto llame al método de otro y así sucesivamente. Las clases que se deben crear recrean un sistema N-Tier y se listan a continuación. Todas las mismas tienen un sólo método llamado *Ejecutar* el cual instancia y ejecuta el método del nivel N+1.

- Nivel 1: clase *CapaVista*
- Nivel 2: clase *CapaControlador*
- Nivel 3: clase *CapaAplicacion*
- Nivel 4: clase *CapaDominio*
- Nivel 5: clase *CapaPersistencia*

En la clase *CapaPersistencia* lance la excepción *ErrorPuntualException*, incorporando en su constructor, además del mensaje, la fecha y hora del error.

La excepción no debe propagarse más allá de la clase *CapaAplicacion* en la cual se debe tratar la excepción lanzada, lanzando una *CapaAplicacionException* incorporando la anterior como origen. De esta forma un subsistema puede enmascarar los errores sucedidos.

En el método de la clase *CapaVista* capture la última excepción lanzada, imprimiendo por pantalla los datos de las dos excepciones lanzadas, el callstack y otros datos que considere. En este punto no se debe tener ninguna referencia a la excepción *ErrorPuntualException*.

Ejercicio 2

Desarrolle los componentes necesarios para proveer la configuración de una aplicación, los cuales estarán en formato JSON. De momento, la aplicación solamente requiere un archivo donde se configure lo necesario para interactuar con una API RESTful, pero se espera que a futuro se soporten más archivos con el menor impacto sobre el código existente y maximizando la reusabilidad del código.

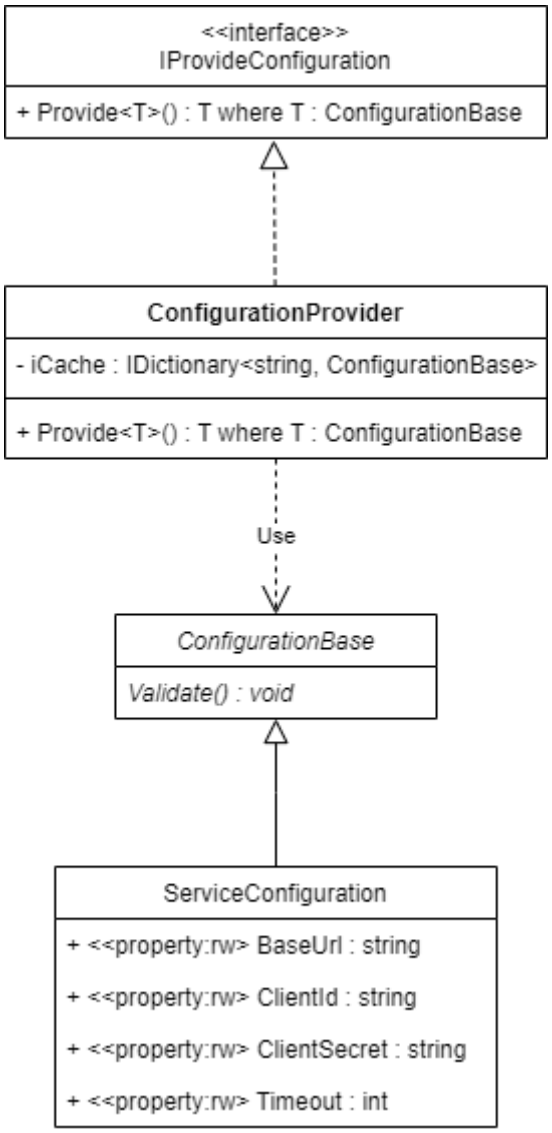
El contenido del archivo de configuración es el siguiente:

```
{
```

```
  "BaseUrl": "https://test.api.com",
  "ClientId": "0RiotGMf45lYSmoD",
  "ClientSecret": "zsOJpfOR5f26DPLU",
  "Timeout": 3000
}
```

Las propiedades *BaseUrl*, *ClientId* y *ClientSecret* son obligatorias y en el caso de *BaseUrl* debe ser una URL válida. La propiedad *Timeout* es opcional, y en el caso de no proveerse un valor se deberá asumir el valor 3000.

El diagrama de clases es el siguiente:



Se brindan los siguientes lineamientos:

- La clase abstracta *ConfigurationBase* define únicamente la operación abstracta

Validate, la cual debe comprobar que la configuración sea correcta y en el caso de que no lo sea debe lanzar una excepción.

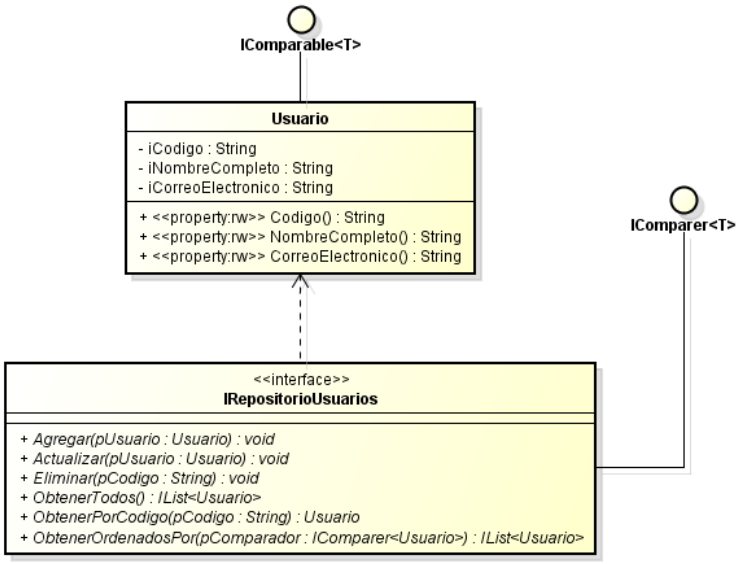
- La clase *ServiceConfiguration* sirve de modelo para el archivo de configuración del servicio, y debe implementar el método *Validate* de acuerdo a lo especificado previamente.
- La clase *ConfigurationProvider* será la encargada de proveer la configuración correspondiente, de acuerdo a los siguientes lineamientos:
    - Si bien actualmente solo está presente el modelo *ServiceConfiguration*, la clase debe ser genérica para cualquier modelo que extienda de *ConfigurationBase*, por lo que no debe contener referencias a ningún modelo en concreto.
    - El archivo de configuración a leer debe coincidir con el nombre del modelo de configuración del archivo con la extensión JSON. Por ejemplo, el archivo correspondiente al modelo *ServiceConfiguration* se llamará *ServiceConfiguration.json*.
    - Los archivos solo se recuperarán del disco si es que no están en caché. Para ello se utiliza el atributo *iCache*, donde la clave es el nombre de la clase del modelo y el valor la instancia de *ConfigurationBase* del modelo correspondiente.

Para leer un archivo JSON, utilice la librería *Newtonsoft* y tome como referencia el código disponible en la ayuda de la misma:
https://www.newtonsoft.com/json/help/html/ReadJson.htm

Escriba los tests unitarios correspondientes para probar el comportamiento de las distintas clases.

Ejercicio 3

Considere la siguiente interface *IRepositorioUsuarios*, correspondiente a un patrón de diseño Repository.



Se pide lo siguiente:

- Investigue conceptos y usos del mencionado patrón.
- Realice una implementación en memoria del repositorio, es decir que deberá crear una clase que implemente dicha interface y mantenga instancias de la clase *Usuario* en una colección. Utilice la implementación de *IDictionary<K, V>* que crea conveniente para manejar mediante pares clave – valor accesos aleatorios a los objetos almacenados, donde el código sea la clave y la instancia de *Usuario* sea el valor.
- Mediante la interface *IComparable<T>* establezca que el orden por defecto de las instancias de *Cliente* sea alfabéticamente por código.
- Implemente al menos tres criterios de ordenamiento para utilizar con *IRepositorioUsuarios.ObtenerOrdenadosPor*.