Universidad Tecnológica Nacional Facultad Regional Avellaneda										
Técnico Superior en Programación - Técnico Superior en Sistemas Informáticos										
Materia: Laboratorio de Programación II										
Apellido:			Fecha:	05/12/2023						
Nombre:			Docente(2):							
División:	2ºC		Nota(2):							
Legajo:			Firma(2):							
Instancia(1):	**PP**		**RPP**		**SP**		**RSP**	X	**FIN**	

(1) Las instancias validas son: 1er Parcial (**PP**), Recuperatorio 1er Parcial (**RPP**), 2do Parcial (**SP**), Recuperatorio 2do Parcial (**RSP**), Final (**FIN**). Marque con una cruz.
(2) Campos a ser completados por el docente.

IMPORTANTE:

- **2 (dos) errores en el mismo tema anulan su puntaje.**
- La correcta documentación y reglas de estilo de la cátedra serán evaluadas.
- El proyecto debe ser creado en .Net 5.
- Colocar sus datos personales en el nombre de la carpeta principal y la solución: Apellido.Nombre.Div. Ej: Pérez.Juan.2D. No sé corregirán proyectos que no sea identificable su autor.
- No se corregirán exámenes que no compilen.
- **Reutilizar** tanto código como crean necesario.
- Colocar nombre de la clase (en estáticos), **this** o **base** en todos los casos que corresponda.
- Aplicar los principios de los 4 pilares de la POO.

---

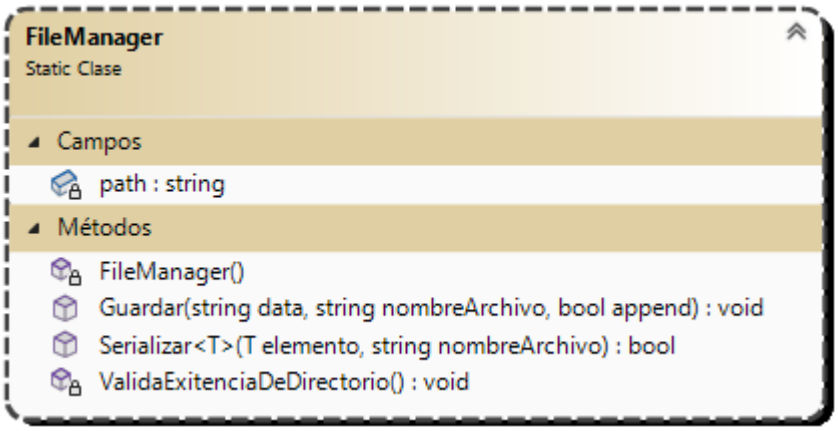
*TIEMPO MÁXIMO PARA RESOLVER EL EXAMEN **120** MINUTOS.*

---

1. Partir de la solución entregada. Modificar su nombre con el siguiente formato: [APELLIDO].[NOMBRE].
2. Implementar la BD desde el backup enviado.

Files

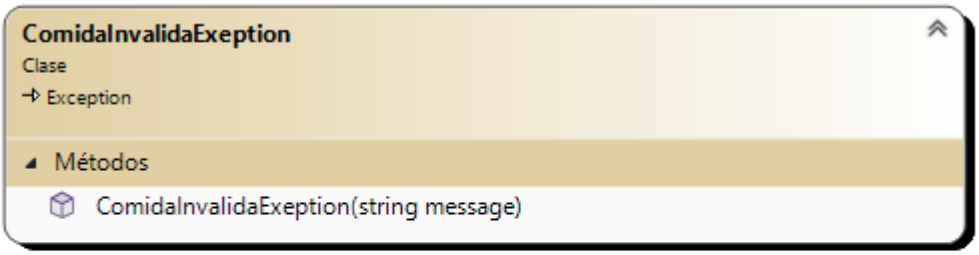
3. Dentro del proyecto se deberá respetar el siguiente esquema:

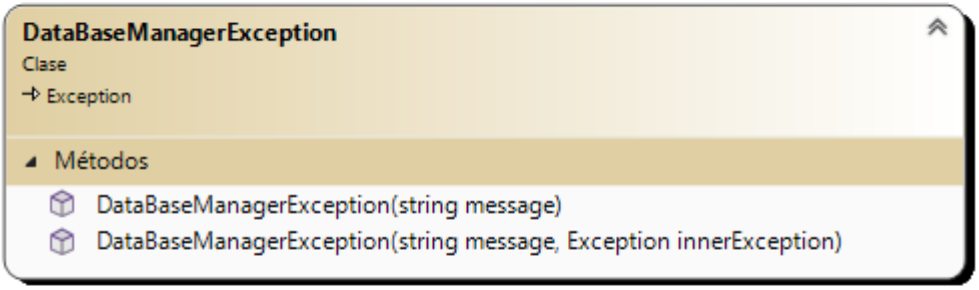


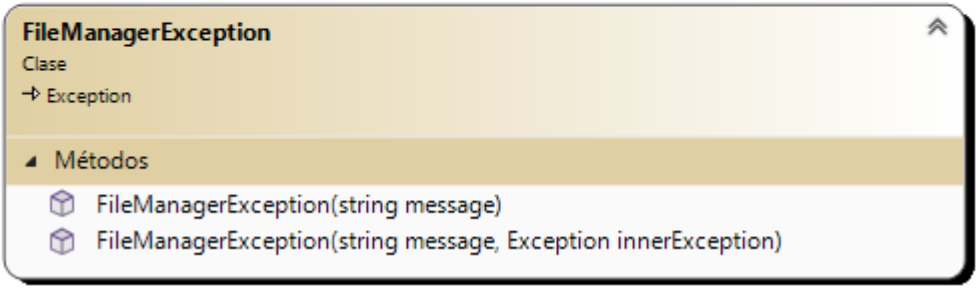
4. FileManager será estatica.
    a. En el constructor:
        i. En el atributo path (privado) se almacenará la referencia al escritorio de la pc. Y se le concatenara un el nombre de la carpeta del parcial: ej {path escritorio}+`\\20231207_Alumn\\`
        ii. Llamar al método ValidaExistenciaDeDirectorio.
    b. ValidaExistenciaDeDirectorio:
        i. Sera privado.
        ii. Si no existe el directorio almacenado en path, se creará.
        iii. En caso de producirse una excepción al momento de la creación, esta deberá ser capturada y relanzada en una nueva excepción denominada FileManagerException, la cual contendrá el mensaje: “Error al crear el directorio”.
    c. Guardar:
        i. Sera el método para poder generar archivos de texto. El mismo se podrá usar para agregar información a un archivo ya existente o sobre escribirlo.
    d. Serializar:
        i. Sera genérico y solo aceptara tipos por referencia.
        ii. Sera el método encargado de serializar en json.
        iii. Retornara true al terminar la serialización;
    e. Reutilizar codigo donde crea necesario.

Excepciones

5. Dentro del proyecto respetar el siguiente esquema:





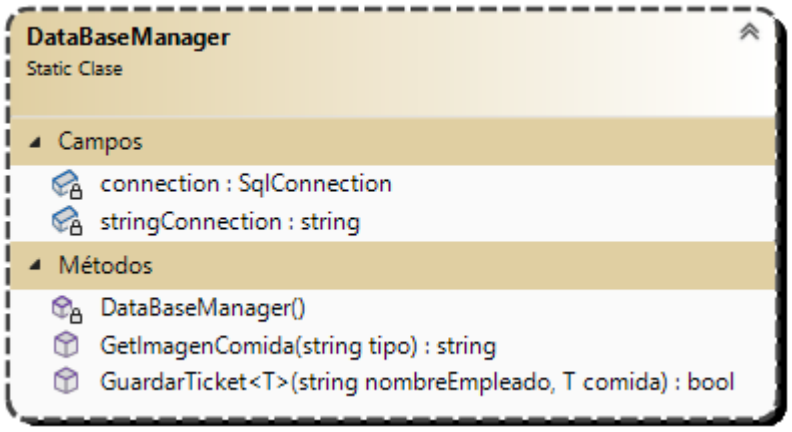


6. Controlar las posibles excepciones producidas y almacenarlas en un archivo txt denominado logs.txt

Bases de datos

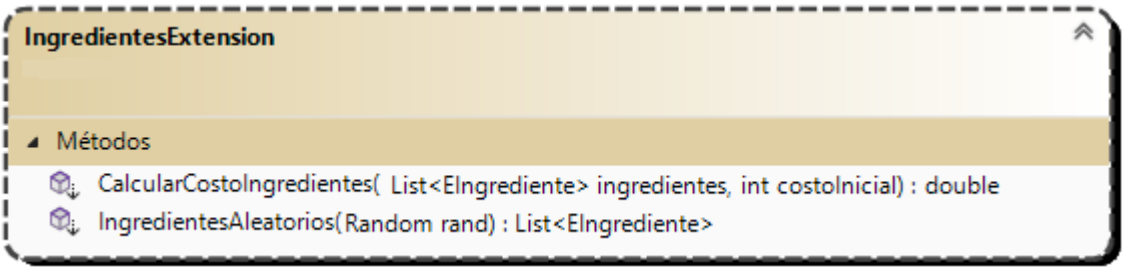
7. Dentro del proyecto se deberá respetar el siguiente esquema:

8. DataBaseManager será estática:



   a. En el constructor de clase inicializar el string connection.
   b. ***GetImagenComida***, recibirá el tipo de comida, el cual usará para filtrar en la tabla **comidas**. Retornara la columna 2 (contando desde cero), la cual corresponder a la URL de la imagen almacenada en la BD. En caso de que no exista el tipo de comida se lanzara una excepción ***ComidaInvalidaException***.
   c. ***GuardarTicket***, será genérico, solo aceptará tipos que implementen la interfaz IComestible y tengan un constructor publico sin parámetros. Recibirá el nombre del cliente y la comida. En la tabla **tickets** almacenara el nombre del empleado y el ticket de la comida.
   d. Cualquier excepción que se pueda producir se deberá encapsular en una nueva denominada ***DataBaseMangerException***, haciendo referencia al error producido (“leer” o “escribir”)

Métodos de extensión



9. ***CalcularCostoIngrediente*** extenderá la clase `List<EIngrediente>` la cual, adicionalmente recibirá un valor como parámetro, el cual se denomina costo inicial. Su tarea será, tomar el costo inicial e incrementar su valor porcentualmente en base a los valores de la lista de Eingredientes. Retornara este valor incrementado.

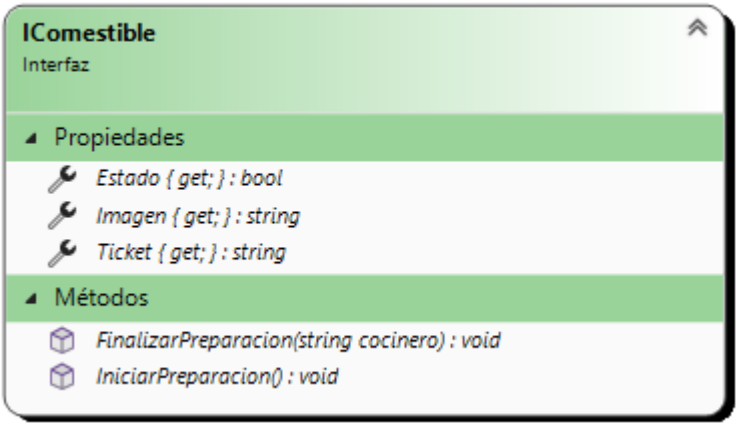
10. ***IngredientesAleatorios*** extenderá la clase `Random`. Su tarea será generar una lista de ingredientes ej:

```
List<EIngrediente> ingredientes = new List<EIngrediente>()
      {
          EIngrediente.QUESO,
          EIngrediente.PANCETA,
          EIngrediente.ADHERESO,
          EIngrediente.HUEVO,
          EIngrediente.JAMON,
      };
```

Generará un numero random desde 1 hasta el largo de la lista + 1. Y retornara una lista de ingredientes en base a la cantidad obtenida de forma aleatoria. Usar el siguiente método: `return ingredientes.Take(“acá va el numero aleatorio”).ToList();`

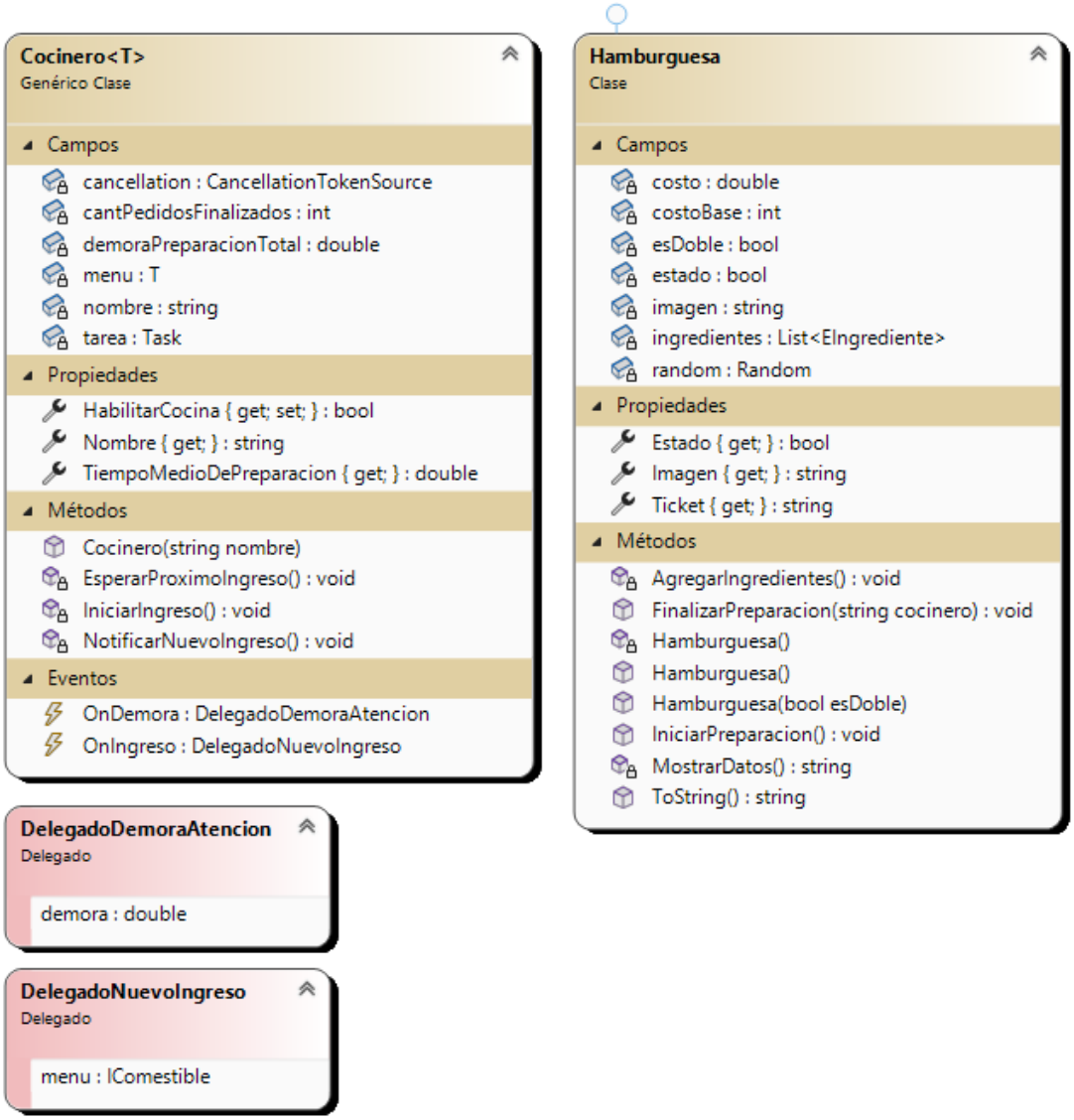
Interfaces

11. Dentro del proyecto se deberá respetar el siguiente esquema:



Entidades

12. Dentro del proyecto se deberá respetar el siguiente esquema:



13. **Hamburguesa**, deberá implementar IComestible.
    a. ***AgregarIngredientes***, agregara los ingredientes de forma aleatoria.

b. ***IniciarPreparacion***, si el estado es **false,** generara un numero aleatorio de 1 hasta 9 y asignara la imagen de la hamburguesa, para ello se le deberá enviar al método que obtiene la imagen el siguiente string: `$"Hamburguesa_{“Acá va el numero aleatorio”}".` Luego llamara a agregar ingredientes.
c. ***FinalizarPreparacion***, asignará el costo a la hamburguesa, este será en relación al costo base y los ingredientes de la hamburguesa. Luego cambia el estado de la hamburguesa.

14. **Cocinero**, será genérica, solo podrá recibir tipos que implementen la interfaz **IComida** y posean un constructor publico sin parámetros:
15. La propiedad HabilitarCocina:
    a. El GET retornara True, si la tares no es nula y estado de la tarea es Running o WaitingToRun o WaitingForActivation.
    b. En el SET, si el valor recibido es TRUE y la tarea es nula o su estado no es Running o no es WaitingToRun o no es WaitingForActivation, se instanciará un nuevo CancelationTokenSource y se llamará a InciarIngreso. De lo contrario se llamará al método Cancel de cancellation.
16. El método ***IniciarIngreso*** será privado y:
    a. Ejecutara en un hilo secundario la acción de que:
        i. Mientras no se requiera cancelación de la tarea de:
            1. Invocara al mensaje NotificarNuevoIngreso y
            2. EsperarProximoIngreso.
            3. Incrementar cantidad de pedidos finalizados en 1.
            4. Guardar ticket en la BD.
17. El método ***NotificarNuevoIngreso***, verificara si el evento OnIngreso posee suscriptores y en caso exitoso realizara:
    a. Instanciara un nuevo menú
    b. Iniciar la preparación del menú.
    c. Notificara el menú.
18. El método ***EsperarProximoIngreso*** si posee un suscriptor notificara los segundos transcurridos mientras que (Utilizar Thread.Sleep para dormir el hilo 1 segundos antes de ir decrementando):
    a. El hilo secundario no requiera cancelación.
    b. El estado del pedido **false**.
    c. Al finalizar incrementar el valor de demoraPreparacionTotal en base al tiempo transcurrido.
19. La propiedad ***TiempoMedioDePreparacion*** retornara el resultante de dividir la demora en preparación total sobre la cantidad de pedidos finalizados

Formulario

20. Desarrollar todo lo indicado con comentario //Alumno:

Test Unitarios

21. Darle un nombre claro al proyecto, sus clases y sus métodos
22. Agregar 2 test unitarios:
    a. Forzar, mediante el código la ejecución de FileManagerException, validar que suceda de forma correcta.
    b. Al instanciar un nuevo cocinero, la cantidad de pedidos finalizados debe ser igual a 0 (cero).

Al finalizar, colocar la carpeta de la carpeta de la Solución completa en un archivo ZIP que deberá tener como nombre Apellido.Nombre.division.zip y compartir este por Discord sólo con el docente titular de la cursada.