Trabajo Práctico 3 - Colecciones y Responsabilidades

Este trabajo práctico ataca los dos últimos temas del cuatrimestre.

*Por un lado presenta problemas relacionados con el uso de **colecciones de datos**. Estas serán las primeras estructuras múltiples que utilizaremos (más adelante conoceremos otras).*

*Esto nos permitirá trabajar en problemas con una estructura funcional y de datos mayor, en donde aparece otra faceta a tener en cuenta: la **distribución de responsabilidades**.*

Estos ejercicios intentan darnos una acercamiento a ambos temas, los que son aparentemente distintos y desconectados pero nos ayudarán a entender dónde y cómo encontrar soluciones a problemas cada vez más interesantes, combinando clases y distribuyendo las tareas con el objetivo de que trabajen en conjunto.

Nota: Los siguientes ejercicios te dan varios ejemplos de cómo combinar clases. Al resolverlos evitá exponer los atributos de tipo objeto de cada clase para respetar las responsabilidades de cada una de ellas.

Ejercicios

Ejercicio 1. Agenda Personal [En clase]

Para desarrollar las funcionalidades solicitadas en este ejercicio debes incorporar y modificar la estructura de la clase **Persona** del TP2 (Introducción a POO) y agregarle el atributo dni, de solo lectura, que servirá como campo identificador de cada persona. Además, agregale un constructor parametrizado que inicialice todos los atributos a partir de los valores recibidos por parámetro.

Definí la clase **Agenda**, la que tiene una colección de personas (**ArrayList<*Persona*>**) y los siguientes métodos:

- Constructor público que instancia vacía la colección de personas (el ArrayList).
- Método privado `buscarPersona()` que busca en la colección la persona con el dni recibido por parámetro. Si la encuentra la devuelve, sino devuelve *null*.
- Método público `agregarPersona()` que recibe por parámetro todos los datos necesarios para registrar una persona. Si no existe la persona con el dni recibido crea una nueva instancia de **Persona** con todos los datos recibidos y la agrega a la colección. Devuelve un booleano indicando si la acción se pudo realizar (si se agregó la persona) o no.
- Método público `removerPersona()`, que recibe como parámetro el número de dni de una persona. Si encuentra en la colección una persona con el

mismo dni recibido la extrae de la colección y la devuelve. Cuando no la encuentra devuelve null.

- Método público `modificarDomicilio()` que recibe por parámetro un dni y un domicilio. Si encuentra en la lista a la persona con ese dni modifica su domicilio con el recibido por parámetro. Retorna un booleano indicando si la operación fue exitosa o no.
- Método público `listarPersonas()` que emite por pantalla en formato de listado a cada persona encontrada en la colección (con sus respectivos atributos).
- Método público devolverUltimo() que no recibe parámetros y devuelve el último elemento de la lista (si es que ésta contiene elementos).
- Método público eliminarTodosElementosAMano() que no recibe parámetros y elimina todos los elementos de la lista (sin utilizar el método clear).

Crear la clase **Test** con su función ***main()***. En esta función se creará una instancia de la clase **Agenda** y se pide, por lo menos, comprobar el correcto funcionamiento de los 4 métodos a desarrollar citados previamente en los siguientes casos: agregar 3 personas, eliminación no exitosa, eliminación exitosa, modificación de dirección no exitosa, modificación de dirección satisfactoria y listado de personas.

Ejercicio 2. Listado de nómina completo [en grupo]

Una compañía está compuesta por distintas áreas, las que a su vez están formadas por una o más oficinas. Los empleados se asignan a una única oficina y no se comparten.

La compañía precisa informar el nombre de todos sus empleados, indicando a qué área y a qué oficina pertenecen.

¿Qué métodos agregarías, y en qué clases?

Con esto, generá el listado de empleados de toda la empresa, sector por sector (con su encabezado) y oficina por oficina (también con su encabezado).

Ejercicio 3. Las casas de mi barrio [tarea]

Una inmobiliaria posee una lista con todos los barrios en donde tiene propiedades a la venta. Cada barrio tiene su nombre y una lista con todas las propiedades a vender. De cada propiedad se sabe el domicilio (String), su precio y su tipo (Departamento, Casa o PH). Basándose en el esto:

A. Diseñá el diagrama de clases que lo modelice, con sus relaciones, atributos y métodos.
B. Explotá el método `mostrarPropiedades(...)` de la clase **Barrio**, que reciba un tipo de propiedad y muestre las direcciones y precios de todas las propiedades de ese tipo.
C. Explotá el método `mostrarPropiedades()` de la clase **Inmobiliaria**, el cual debe mostrar la dirección y el precio de todas las propiedades que posea a la venta.
D. Desarrollá el método `obtenerBarrioMaxProp()` de la clase **Inmobiliaria**, que devuelva una lista de él o los barrios con la mayor cantidad de

propiedades en cartera. Debe ser usado desde el método de la misma clase `mostrarBarrioMaxProp()`.

E. Una propiedad puede ser cargada en el barrio incorrecto. Explotá el método `cambiarPropiedadDeBarrio()` de la clase **Inmobiliaria**, que reciba un domicilio y un barrio y mueva la propiedad asociada a ese domicilio al nuevo barrio, removiéndola del barrio anterior.

F. Explotá el método `borrarPropiedad()` de la clase **Inmobiliaria**, que reciba un domicilio y elimine la propiedad asociada a ese domicilio del barrio donde se encuentre.

Completá el programa para probar los métodos pedidos.

Ejercicio 4. Paseando a Pichichus [En clase] [FPR]

Juan es dueño de los perros Pichichus y Sultán, aunque podría tener más. Juan pasea a Pichichus por la mañana y a Sultán por la tarde. Independientemente de eso, ambos perros están entrenados para, eventualmente, pedir que los saquen a hacer sus necesidades.

Cada perro reconoce a una única persona como su dueño y responde al llamado por su nombre (cuando el dueño lo busca). Para salir el perro debe usar su collar. Cuando esto ocurre el perro comienza a mover la cola de alegría.

El collar de cada perro queda colgado en un perchero (este puede guardar hasta tres collares) hasta que el dueño lo toma para ponérselo al perro que llevará de paseo. Para saber de cuál perro pertenece, cada collar tiene una chapita identificatoria con el nombre del perro.

En base a lo enunciado implementá lo siguiente:

Para preparar el paseo Juan siempre realiza las mismas acciones:

A. Busca el collar del perro que va a pasear en el perchero.
B. Llama al perro que va a sacar a pasear.
C. Cuando el perro ya tiene puesto el collar salen a pasear.
D. Siempre al regresar del paseo el collar vuelve a quedar en el perchero.

Por su lado, el perro se pone contento y lo demuestra moviendo la cola cuando tiene el collar puesto. Si por algún motivo Juan no encontrara el collar del perro que va a pasear o el perro no respondiera a su llamado el paseo no se realiza.

Nota 1: en este programa, que el perro "no responda al llamado" sería implementado como que el perro que se busca dentro de los perros asociados a este dueño, no existiera.

Nota 2: ¿Cómo harías que uno de los perros le pidiese a su dueño que lo saque a pasear?

Ejercicio 5. Biblioteca [En clase]

Una biblioteca tiene una vasta colección de libros. Cada libro tiene un título, un género (novela, teatro, poesía, ensayo u otros), editorial, año y autor. De cada libro

puede haber más de un ejemplar y cada ejemplar tiene un identificador, una fecha de préstamo, y un estado (en BIBLIOTECA, PRESTADO o EN_REPARACION).

Las editoriales tienen nombre y país, y los autores tienen nombre, nacionalidad y fecha de nacimiento.

De los lectores se sabe su dni, su nombre, su dirección y su teléfono. Cada Lector puede tener hasta un máximo de 3 préstamos vigentes. Cuando se realiza el préstamo de un ejemplar se registra la fecha de retiro y el ejemplar que se está retirando. El préstamo expira a los 7 días, y por cada día de retraso se impone una "multa" de dos días sin posibilidad de retirar un nuevo libro.

Tomando en cuenta el enunciado descripto, realizar:

a) El diagrama de clases correspondiente.
b) La explotación del método *prestar(...)* de la clase ***Biblioteca***, más todos los algoritmos derivados que pudieran surgir. Este método recibe la instancia del ***Libro*** requerido y del ***Lector*** que lo pide, y devuelve un valor *enumerado* con los siguientes posibles valores y significados:
    i. **PRESTAMO_EXITOSO**: El libro es prestado exitosamente (en este caso, antes de salir del método hay que marcar el ***Ejemplar*** como prestado, entregándoselo al lector y procesar y registrar el ***Prestamo***).
    ii. **NO_HAY_EJEMPLARES**: No hay ejemplares disponibles del libro.
    iii. **TOPE_PRESTAMOS_ALCANZADO**: El lector ya posee tres libros en préstamo.
    iv. **MULTA_VIGENTE**: El lector posee una multa en vigencia.
c) A partir del ejercicio incompleto que encontrarás en el proyecto asociado a esta práctica (que debes descargar desde el Aula Virtual) completá el mismo método en Java. Probá el programa asegurándote de que los resultados sean los correctos.

Ejercicio 6. ORTFlix [En grupo]

ORTFlix es un nuevo sitio en el cual se pueden ver películas online. Para diferenciarse de su competencia, los clientes de ORTFlix pueden optar por dos tipos de servicios: el servicio ***Standard*** y el servicio ***Premium***. El servicio Premium da acceso a películas más nuevas (generalmente estrenos de ese mismo año).

Del cliente del cual se conocen su DNI, su nombre y el saldo a pagar que debe (si no debe estará en cero, si pagó de más será negativo).

Para ver cualquier película, el cliente no debe poseer deuda, y solo podrá ver contenido Premium si está suscripto al servicio Premium.

Existe un método de la clase Cliente llamado *esDeudor()* que devuelve verdadero en caso que posea deuda.

Por último, al sitio le interesa llevar un historial de las películas que vió cada cliente.

Basado en este enunciado, realizá el diagrama de clases que lo modelice, con sus relaciones, atributos y métodos. Además explotá los métodos `verPelicula()`, el

cual recibe como parámetros el DNI de un cliente y el nombre de una película. Debe devolver alguno de los siguientes resultados:

- CLIENTE_INEXISTENTE: Dni no encontrado.
- CONTENIDO_INEXISTENTE: La película no existe.
- CLIENTE_DEUDOR: El cliente posee deuda.
- CONTENIDO_NO_DISPONIBLE: La película es Premium y el cliente no está abonado a dicho servicio.
- OK: La película puede verse sin inconvenientes. La película debería quedar registrada en el historial de películas vistas del cliente.

Desarrollá la explotación del método *darDeBaja(...)*. Este método recibe un DNI y elimina de la lista al cliente con el mismo DNI. Si al darlo de baja el cliente es deudor se lo agregará a una lista negra (por ejemplo, para no volver a asociarlo sin cancelar la deuda previa).

Desarrollá el método *darDeAlta(...)*. Este método recibe un DNI y la categoría y agrega al cliente en su lista de clientes. Debe devolver alguno de los siguientes resultados:

- CLIENTE_EXISTENTE: Dni ya encontrado.
- CLIENTE_DEUDOR: El cliente posee deuda previa y se encuentra en la lista negra.
- ALTA_OK: Están dadas las condiciones para dar de alta al cliente.

Desarrollá el método *depurarListaNegra(...)* que recibe un importe tope y elimina de la lista negra todos los ex-deudores con un importe de deuda menor o igual al recibido por parámetro.

Completá el programa para probar los métodos pedidos.

Ejercicio 7. MusicApp [Tarea]

Nos solicitan desarrollar una aplicación para escuchar música. La misma cuenta con una lista de usuarios y otra de canciones.

Sabemos que hay tres categorías de **usuario**: GRATUITO, REGISTRADO y PREMIUM. Los usuarios con licencia gratuita sólo pueden escuchar hasta cincuenta canciones por día y que no sean de la categoría restringida (estrenos, cortes especiales, etc,), los usuarios estándar no pueden escuchar las canciones restringidas pero son libres de escuchar todas las canciones que quieran y los usuarios premium pueden escuchar cualquier canción sin límites.

De los objetos de tipo **Cancion** conocemos su nombre (String), autor (String) y el género musical (Rock, Pop, Clásica, Trap, Cumbia, etc.). Además, cada canción tiene dos contadores: uno llevará la cuenta de los días de publicación y otro la cantidad de veces que la canción fue escuchada. Toda canción estará restringida para acceso sólo premium mientras no haya sido escuchada más de mil (1000) veces o hayan pasado siete (7) días desde su publicación.

Vale aclarar que dentro de la lista de canciones no se puede repetir la combinación *nombre* y *autor* (campos de identificación).

De cada Usuario sabemos el email (String, será el campo que lo identifique), su apellido (String), su edad (int) y su categoría de usuario (GRATUITO, ESTANDAR, PREMIUM). Además, contamos con una lista de canciones que el usuario ya escuchó. Basado en esto, se requiere.

A. Diseñar el diagrama de clases distribuyendo los métodos que aparecen a continuación y completando con lo que haga falta.
B. Desarrollar los siguientes métodos en las clases que correspondan:
    a. Método privado `buscarCancion()`: recibe por parámetro un nombre y un autor. Retorna un objeto de tipo Cancion si la encuentra en el catálogo general de canciones, o *null*.
    b. Método privado `buscarUsuario()`: recibe por parámetro una dirección de email. En caso de encontrarlo en el listado de usuarios lo devuelve, de lo contrario devuelve *null*.
    c. Método público `altaCancion()`: Recibe los datos necesarios para dar de alta una canción (sólo si no fue cargada previamente) en la lista de canciones. Devuelve un booleano que indique si la canción fue agregada.
    d. Método público `altaUsuario()`: Recibe los datos necesarios para dar de alta un usuario (si no existe desde antes) en la lista de usuarios. Devuelve un booleano indicando si la operación fue exitosa o no.
    e. Método público `escucharCancion()`: Recibe email, nombre de canción y autor. Si el usuario y la canción existen agrega la canción a la lista de canciones ya escuchadas por este usuario (esta lista soporta repetidos). Devuelve un valor *enum* que indica el resultado de la operación. Los valores a devolver pueden ser:
    CANCION_ESCUCHADA, cuando la acción es exitosa.
    USUARIO_INEXISTENTE, cuando no se encuentra el usuario con el email recibido.
    CANCION_INEXISTENTE, cuando no se encuentra la canción.
    LIMITE_ALCANZADO, cuando el usuario es GRATUITO y se alcanzó el límite de la categoría.
    CANCION_NO_DISPONIBLE, sólo se puede escuchar cuando el usuario es PREMIUM. Una canción es restringida cuando no supera los siete (7) días de publicación o las mil (1000) escuchas.
    f. Método público `listarUsuarios()`: Muestra un listado con los usuarios registrados en la aplicación más la cantidad de canciones que escuchó cada uno.
    g. Método público `listarCancionesPorUsuario()`: Recibe por parámetro el mail de un usuario, y de estar registrado en la aplicación, muestra un listado con las canciones que escuchó hasta el momento y devuelve un mensaje de operación exitosa. Caso contrario, emite por pantalla un mensaje que indica que el usuario es inexistente.

Desarrollá los métodos auxiliares privados que consideres necesarios para el funcionamiento del programa. Recordá proteger el encapsulamiento de los objetos.

Crear una instancia de la clase Aplicación en la función *main()* de la clase **Test** para probar todos los métodos públicos definidos.

Ejercicio 8. Los turnos del Doctor Sueño [Tarea]

El Dr. Sueño quiere un programa que le permita manejar la agenda de turnos del día siguiente. Los mismos se asignan por orden de llegada, pero solo otorga una cantidad limitada de números, la cual se debe pasar por parámetro al crear la agenda.

El programa debe permitir la registración de los turnos de los pacientes. De cada Paciente se conocen los siguientes datos: dni (identificador), nombre, apellido, y teléfono.

Además de contener la información de quien se inscribió en la consulta, cada Turno cuenta con un atributo de presencia (booleano, que indica si el paciente se presentó al turno que reservó o no).

Se pide crear los siguientes métodos en las clases que correspondan:

A. Método privado `buscarTurno()`: Devuelve, si existe, un turno de la lista de turnos dado un DNI del paciente recibido por parámetro. De lo contrario retorna null.
B. Método público `registrarTurno()`: Dado todos los datos de un paciente, crea un Turno para el mismo y lo agrega a la agenda de turnos. Además, retornará como resultado la confirmación del registro. Tener en cuenta que si se cumple alguno de los siguientes casos, el turno no se agregará a la lista, retornando el respectivo estado con un enumerado que refleje estas posibilidades:
    - El médico no tiene más turnos.
    - El paciente (dni) ya tenía previamente un turno asignado en la lista.

   En caso de que no se cumpla alguna de estas dos condiciones devolver un tercer valor que signifique *Turno Confirmado*.
C. Método público `listarTurnosAsignados()`: Mostrará por pantalla la cantidad de turnos asignados hasta el momento y los datos de cada turno asignado. (Incluye visualizar los datos de los pacientes.)
D. Método público `darPresente()`: Dado un dni recibido por parámetro, si existe un turno de algún paciente en la lista con ese dni lo marca como presente. Retorna un booleano indicando si pudo o no realizar la acción.
E. Método público `anularTurno()`: Dado un dni recibido por parámetro, quita el turno de la lista si encuentra al paciente. Retorna un booleano indicando si pudo o no realizar la acción.
F. Método público `obtenerAusentes()`: Retorna un ArrayList de elementos Paciente, con aquellos/as pacientes que no se presentaron para los turnos que solicitaron.

Escribir un programa que instancie la **AgendaMedica** y pruebe todos los métodos públicos definidos.

Ejercicio 9. Preparando la receta [En clase]

En la casa inteligente de Ricardo todos sus artefactos son capaces de comunicarse entre sí. A Ricardo le encanta la cocina y no hay receta que no sea capaz de preparar pero es muy desordenado al momento de armar su lista de compras. ¡El problema es que a menudo olvida comprar los ingredientes!

Por esta razón decidió comprarse un recetario electrónico capaz de comunicarse con su heladera y su alacena, ambos Depósitos inteligentes. La diferencia entre la heladera y la alacena es que la primera está refrigerada.

De cada producto que hay en la alacena y en la heladera se sabe su nombre, cantidad, y fecha de vencimiento.

Cada receta del recetario tiene un código, su nombre y una lista con los ingredientes necesarios para la preparación de una porción, indicando la cantidad necesaria y si dicho ingrediente es refrigerado o no.

Debes ayudar a Ricardo con su nuevo recetario desarrollando;

1. El diagrama de clases que representa este escenario.
2. El método `prepararListadoAComprar(....)` de la clase recetario el cual recibe por parámetro el nombre de la receta a cocinar, la cantidad de porciones deseadas y un depósito. El método devuelve una lista con los nombres de los ingredientes que faltan y la cantidad que hace falta comprar para que Ricardo pueda prepararla. Esta lista puede quedar vacía (si no hace falta comprar nada) y deberá ser completada buscando los ingredientes en cada depósito dependiendo de si el ingrediente es un producto refrigerado o no refrigerado.
3. El método `recetasPosiblesAPreparar(....)` el cual recibe como parámetro un Depósito y un entero indicando la cantidad de porciones deseadas y debe devolver una lista con el o los nombres de la Recetas posibles que se pueden cocinar con los ingredientes que se tienen.

Completá el programa con la clase Test y en su *main()* escribí lo necesario para probar los métodos pedidos.

Ejercicio 10. Buscando candidatos [Tarea]

Las empresas suelen acceder a las bolsas de trabajo de instituciones educativas orientadas a su rubro para encontrar posibles candidatos a contratar. Para esto las instituciones cuentan con la lista de sus alumnos que están cursando la carrera y quieren participar de la bolsa de trabajo. De cada alumno se sabe su nombre, su email y todas las materias que tiene aprobadas. De cada materia se sabe su nombre y la nota final obtenida.

Para que un candidato sea considerado en una búsqueda debe tener un mínimo de 5 materias aprobadas. La empresa que busca candidatos establece cuál es el promedio mínimo de notas para que un alumno pueda ser considerado candidato.

Así mismo, por políticas internas de la institución educativa en ningún momento la lista de posibles candidatos que se entrega nunca supera los 20 candidatos.

Por ejemplo, si la empresa está buscando alumnos de promedio 7 o superior, se deberán considerar solo aquellos alumnos que posean al menos 5 materias aprobadas y entre todas las materias que tengan aprobadas su promedio sea de al menos 7. Pero nunca se devolverán más de 20 candidatos, aunque los pudiera haber.

Basado en el enunciado descripto, realizá:

- El UML que lo modelice, con sus relaciones, atributos y métodos.
- La explotación del método `obtenerCandidatos()` de la clase Carrera, que recibiendo como parámetro el promedio mínimo deseado, debe crear y devolver una lista con los nombres y mails de los primeros 20 alumnos que cumplan con los requisitos pedidos. Si no hubiera ningún candidato posible deberá devolver una lista vacía. Si los posibles candidatos no llegarán a ser veinte deberá devolver los que se hayan encontrado.

Nota: Tené en cuenta que **no** se debe devolver una lista de objetos **Alumno** sino una lista que contenga únicamente el nombre y el mail de los alumnos candidatos.
Completá el programa para probar los métodos pedidos. La clase Test, además del main() debe tener un método donde se muestren los candidatos obtenidos en la búsqueda a partir de al menos dos puntajes promedio distintos.

Ejercicio 11. Correcaminos [En grupo]

La empresa “CorreCaminos” se dedica a la organización de carreras de kartings.

De cada carrera se sabe la fecha y hora en la que tuvo lugar, la dificultad de la carrera (PRINCIPIANTE, AVANZADO o AS DEL VOLANTE), la cantidad de vueltas y los pilotos que participaron.

De cada piloto se sabe su nombre y su dni.

Cada piloto lleva registradas las vueltas que realizó. No todos los pilotos completan la totalidad de la carrera y puede ser que deban abandonar antes porque algún problema mecánico, algún accidente, etc. De cada vuelta del piloto se sabe el tiempo obtenido en segundos, y el valor de su vuelta promedio (que se actualiza al cierre de cada vuelta).

Tomando en cuenta el enunciado descripto, realizá:

1. El diagrama UML de clases que lo represente.
2. La explotación del método `buscarCarrera()` de la clase Empresa, que recibiendo como parámetro una fecha, debe devolver una instancia de la Carrera encontrada o null en caso de no existir.
3. La explotación del método `calcularVueltaPromedio()` de la clase Piloto, que debe calcular el promedio de tiempo de dicho piloto entre todas las vueltas que haya realizado.
4. La explotación del método `buscarPilotosPorDebajoDe()` de la clase Carrera y que recibe como parámetro un valor en segundos para devolver una lista de aquellos pilotos que hayan realizado al menos 10 vueltas y cuyo promedio de tiempo no supere los segundos enviados por parámetro.

5. La explotación del método `mostrarMenorPromedio()` de la clase Carrera que no recibe ningún parámetro y debe mostrar por pantalla el nombre y dni del piloto con menor promedio.
6. La explotación del método `pilotosPorCarrera()` de la clase Empresa que no recibe ningún parámetro y debe crear y devolver una lista con la cantidad de pilotos que participaron en cada una de las fecha de las carreras.

Completá el programa para probar los métodos pedidos.

Ejercicio 12. Chequeando los Productos [En grupo]

El chef Cuisine es fenomenal: no hay receta que no pueda inventar a partir de los productos con los que cuenta. El chef es muy exigente y siempre trabaja con productos bien frescos, por lo que necesita revisar permanentemente la antigüedad de los productos almacenados para descartarlos si no están frescos como a él le gusta. Por esta razón nos pidió ayuda para chequear rápidamente esta información.

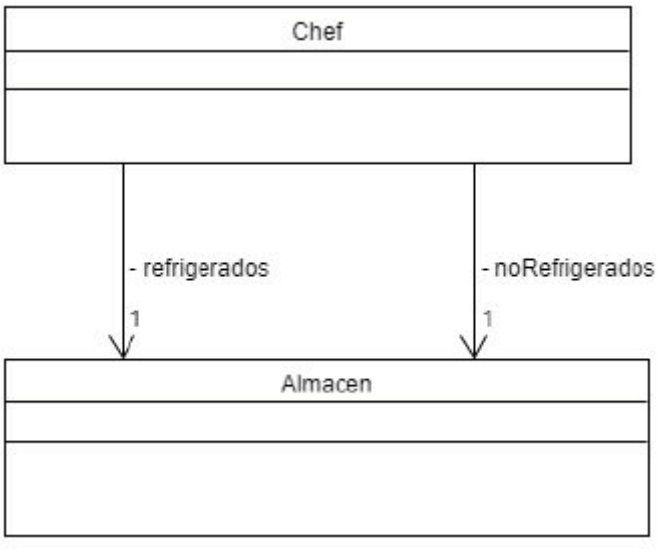
Los productos se guardan por separado en los dos almacenes de la cocina; un almacén para los refrigerados y el otro para los no refrigerados.

Cada producto guardado tiene una descripción o nombre, la fecha de almacenamiento y la cantidad máxima de días que puede estar almacenado y considerarse fresco.

Debemos ayudar al chef desarrollando un método que determine qué productos ya no están frescos y los extraiga de donde están guardados.

Se pide:

- En base al diagrama inicial que acá compartimos, completalo agregando los atributos, métodos y demás elementos (otras clases, enumerados, etc.) que creas conveniente.



- El método `extraerProductosVencidos()` el cual no recibe parámetros y devueve un ArrayList con los productos que se encuentren vencidos sean o no refrigerados. (Posiblemente necesites un método auxiliar que no está definido.)

- Desarrollar el método `chequearProducto(...)` el cual recibe por parámetro el nombre del producto y si es refrigerado o no y debe retornar un Estado con los posibles valores:
  - NO_ENCONTRADO
  - PARA_DESCARTE
  - FRESCO

  Contamos con un método llamado `esFechaMenor(...)` de la clase **Fecha** que pasándole por parámetro una Fecha la compara con el día de hoy y nos devuelve verdadero si es menor al día actual. Puede ser llamada desde cualquier parte que la necesite poniendo `Fecha.esFechaMenor(....)`

Completá el programa para probar que todo funcione correctamente.