Pautas a seguir a partir del esqueleto para resolver el problema

Realizar el diseño UML a partir del enunciado del examen, complétalo con la revisión de las clases que aparecen en el esqueleto

Elimina exclamación roja en proyecto

Si tenemos una **exclamación roja** en el proyecto, hay que revisar build path/configure build path. Seguramente serán problemas de enlace con otros proyectos (por ejemplo con el proyecto útil) o bien que se ha enlazado de manera circular (bucle) con otros proyectos. Eliminar los enlaces y realizarlos de nuevo con los adecuados.

Elimina errores de compilación:

1- **Elimina errores en las listas**. Para eliminar los errores de compilación, empezar por quitar los errores producidos en la declaración de listas. Importar las listas necesarias para eliminar estos errores en todas las clases del proyecto.
2- **Elimina errores en clases que hay que importar** de otros paquetes.
3- **Elimina errores en clases que aún no existen**. Habrá que crear la clase usando siempre quick fix. Antes de crear la clase, indicar en qué paquete crearla. Mirar a ver si ya existe el import para esta nueva clase, porque en ese caso es **el import el que nos indica el nombre del paquete** en el que debe ir esta clase.
4- **Elimina errores en métodos que aún no existen,** creando los métodos a través de quick fix. Más adelante ya se rellenará su contenido.
5- **Elimina errores de gestión de excepciones**. Empezar por gestionar excepción producida al cargar de fichero (FileNotFound). Mirar en el enunciado cómo se desea que sea tratada. Normalmente se transforma en una propia.

Cuando ya se han eliminado todos los errores de compilación:

6- Gestiona los errores de tipo RuntimeException, que se manejan con el manejador por defecto del sistema y provocan la finalización de la aplicación.
7- Implementa las pruebas pedidas
8- Implementa las clases del modelo (atributos y constructor)
9- Implementa métodos de la clase servicio.
    a. Implementa el parser (y todo lo de cargar de fichero). Funcionalidad y gestión de errores
    b. Implementa el serializer (y todo lo de guardar en fichero).
    c. Implementa las ordenaciones. Crear el o los comparadores o el compareTo
10- Implementa operaciones adicionales (crear pedidos en Newsstand, crear notas en ExamMarker, Validar en TransactionProcesor). Controla los errores que el enunciado nos pida, lanzando excepciones propias y recogiéndolas donde sea preciso (en la interfaz si deben acabar o en métodos intermedios si se desea que se continúe).

Importante revisar en el código

1- Asegúrate de que TODAS las listas que se manejan quedan inicializadas **en la propia declaración**.

List<String> myList = new ArrayList<>();

2- Siempre que se devuelva una lista se debe devolver una copia de la misma, y no la propia lista (**copia antes de salir**).

return new ArrayList<String>(myListString);

3- Si se recibe una lista para incorporar sus objetos a una lista propia, deben copiarse sus elementos a la propia lista y no asignarse la referencia externa (**copia antes de entrar**). myListString = new ArrayList<String>(externalList);
4- Si se hacen **búsquedas de objetos** en una lista, debemos **implementar el método equals** en la clase de objetos buscados. Usar creación automatizada por menú y comprobar únicamente los atributos que sea necesario.
5- Si se van a imprimir objetos, será necesario **implementar toString** en el objeto. Mejor **usar StringBuilder** siempre que se concatenen muchos strings.
6- Revisa que se **delegan todas las operaciones posibles** en la clase que almacena los datos para dicha operación.
7- Declara los objetos del tipo de la interfaz, si implementan una interfaz. Por ejemplo List<String> myList en lugar de ArrayList<String> myList. Luego al crearse, ya se crean de un determinado tipo (ArrayList o LinkedList).
8- **Valida** parámetros en TODOS los métodos públicos y constructores.
9- **Casos de Test**. Si el método a probar puede producir excepción, habrá casos para todas las situaciones diferentes por las que pueda saltar excepción (**pruebas de robustez**). Y casos para todas las situaciones que modifiquen el estado del objeto (**pruebas de funcionalidad**).
10- **Implementación de Test de robusted**. Si el método a probar puede producir **excepción**, se recoge la misma con try.. catch. Si hay varias (por ejemplo IllegalArgument y AppException), se recogen todas. Si se implementa un caso en que debe saltar excepción, en el catch se incluye un assertTrue(true) y después de la invocación del método a probar se incluye fail(), puesto que por ahí no debe pasar nunca. Si se implementa un caso en que no debe saltar excepción, en el catch se usa un fail y en el try, después de la llamada, se comprueba que el estado del objeto sea el adecuado, es decir que la funcionalidad esté bien implementada (por ejemplo que la lista generada esté bien, que devuelva el valor adecuado, etc) . Cualquier otro método que no sea el método a probar, si puede producir excepción comprobada, se añade a cabecera.
11- **Implementación Test de funcionalidad**. Si el caso que se implementa, la invocación del método a probar implica modificar una lista interna. Habrá que comparar la lista esperada con la que queda en el objeto tras la invocación (implementar getList si no existe). Al comparar dos listas es necesario que la clase de los objetos a comparar tengan el **método equals comparando todos los campos**.
12- **Creación de lista**. Si usamos Copy.of(..) para crear una lista List<Clase> list = Copy.of(objeto1,objeto2). Hay que tener en cuenta que la lista que se creada es inmutable y por tanto no podemos añadir más objetos a la lista posteriormente. Vale para crear la lista esperada, pero no en otras ocasiones para añadir más objetos.