10.- Un Storage posee a capacidade de almacenar obxectos mediante o método ***store***, eliminalos mediante os métodos **delete** e **remove** e recuperalos mediante **search** e **find**.

- **store**: recibe como argumento o obxecto a almacenar e retorna null ou o obxecto vello si xa estaba almacenado. lanzará unha VerboseException coas mensaxes apropiadas si falla
- **delete**: recibe como argumento o valor dun atributo do obxecto que o identifica de xeito único, e retorna o obxecto eliminado si existe, ou null si non existe. Si se produce un erro se lanza unha VerboseException coas mensaxes apropiadas
- **remove**: recibe como argumento un obxecto que implante a clase Java *Predicate* que se utilizará para decidir que obxectos se van a eliminar. Retorna un array cos obxectos eliminados. Si se produce un erro se lanza unha VerboseException coas mensaxes apropiadas
- **search**: recibe un obxecto que implemente a clase Java Predicate que se utilizará para retornar un array cos obxectos almacenados que cumpren co que establece o Predicate. Si se produce un erro se lanza unha VerboseException coas mensaxes apropiadas
- **find**: recibe como argumento o valor dun atributo do obxecto que o identifica de xeito único, e retorna o obxecto si existe, ou null si non existe. Si se produce un erro se lanza unha VerboseException coas mensaxes apropiadas
- **all:** non ten argumentos. Debe retornar un array con todos os elementos almacenados no Storage

Se pide deseñar Storage facendo uso dos tipos xenéricos (Generics) axeitados para permitir obxectos e chaves de calquera tipo

11a.- Deseñar as clases ***ClientesStorage, ArtigosStorage***, ***VentasStorage*** e ***FacturasStorage*** que son Storage para xestionar o almacenamento de obxectos Cliente, Artigo, Venta e Factura respectivamente. A chave que identifica os clientes é o dni e os artigos o codigo, mentres que as ventas e facturas non son identificadas de xeito único por ningún atributo. Debedes utilizar as estruturas de almacenamento dinámico en memoria máis axeitadas a cada caso (Collections, Maps ... etc) tendo en conta o seguinte:

- Os clientes estarán sempre ordeados por DNI e se poderán recuperar e modificar de xeito óptimo. Non se admiten clientes con DNI duplicado
- Os artigos deben poder recuperarse polo seu código de xeito óptimo. Non se admiten artigos con código duplicado.
- As ventas deben estar sempre ordeadas por data.
- As facturas deben poder recuperarse pola súa posición no Storage.

VentasStorage lanzará unha UnsupportedOperationException si se intenta eliminar ou buscar un elemento por chave de busca.

11b.- Deseñar a clase ***Database***, que constará de cinco atributos estáticos accesibles dende calquera parte que almacenarán respectivamente un obxecto ***ArtigoStock stocks***, ***ClientesStorage tclientes, un ArtigosStorage tartigos, un VentasStorage tventas e un FacturasStorage tfacturas***
Unha vez creada:

- Completar ***VentaBuilder*** de xeito que non se podan facer ventas a clientes non existentes, de artigos non existentes ou de artigos con insuficiente Stock

11c.- Debes engadir a ***Factura*** o método ***void pechaFactura(int num)*** accesible dende calquera parte que calculará e almacenará na factura o importe, ive e total das ventas almacenadas marcando logo a factura como "Pechada" poñendo a ***true*** o atributo pechada e xerando o código de factura a partir do número recibido como parámetro. O código de factura que terá a forma F***AAMMDDDDD***, onde F é a letra F, AA os últimos dous díxitos do ano actual, MM o numero do mes actual en dous díxitos e DDDDD o número recibido como parámetro ocupando 5 díxitos. Non debe ser posible facer modificacións nunha factura "Pechada". Debes comprobar que o código de factura non existe incrementando num si é necesario.
O método ***store*** de FacturasStorage debe pechar a factura antes de almacenala pasándolle como parámetro a posición onde vai ser almacenada, eliminando as ventas correspondentes.

12.- Os ***Printer*** teñen a capacidade de representar obxectos textualmente dos seguintes xeitos:

String[] toCSV(Collection<T> list); Retornará un array de String donde o primeiro elemento é a cabeceira CSV e o resto os detalles.
String toRow(T object); Retornará un array de String cos valores dos campos de object separados por comas
String print(T object); Retornará un String cos formato apropiado para representar en pantalla o obxecto T con toda a información relevante e co formato apropiado.

Se pide a programación de Printer no package do resto das clases e de ClientePrinter, FacturaPrinter, VentaPrinter e ArtigoPrinter dentro do package "principal"

12b.- Crear os métodos estáticos e accesibles dende calquera parte ***Artigo[] fromCSV(String[] csv),Venta[] fromCSV(String[] csv), Cliente[] fromCSV(String[] csv) e Factura fromCSV(String[] csv)*** nas clases Artigo, Venta, Cliente e Factura respectivamente. Estes métodos deben crear os obxectos a partir do String[] csv que os describe no formato empregado polo método toCSV de Printer, lanzando a VerboseException axeitada en caso de erro e tendo en conta que as columnas poden cambiar de posición. **Factura fromCSV so transformará facturas pechadas**