xxx: possibles millores

S'ha prioritzat classes petites i métodes curts, per evitar duplicitat de codi i facilitar el manteniment.
-> Augmenta la clompexitat al enfrentar-se al codi per primer cop.

S'ha intentat posar el menor número de comentaris possibles, intentant que el codi s'expliqui per si sol.

RELACIONS GENERALS

Vistes:
Acoplades només a interfaces de controladors.
Excepte una vista, explicat més endavant.
Opcionalment es permet que les vistes puguin estar acoplades a algunes classes del model, només si aquestes són del tipus Enum.
A priori una vista per a cada funcionalitat / controlador. Si algunes funcionalitats són molt semblants poden compartir la mateixa vista.
El disseny de les vistes s'hauria de repensar si hi hagués una interfaç gráfica (o fer que heredesin de "finestres" comunes), però mentres la interacció sigui per consola no és necessari.

Controladors:
Acomplats només als models i a les interfaces dels Daos (persistencia).

Daos:
Acoplats només als models.

Models:
Només poden estar acoplats a altres models.

FUNCIONAMENT GENERAL, PAQUET FLORISTERIA

Classe principal (App)
0) Té 2 interfaces: Vista + Lògica (que proporciona els controladors)
1) Al arrancar la app es configura en el métode configurarApp (usant classe Configurador)
Tipus de vistes -> vistes per consola
Tipus de controladors -> controladors locals
Tipus de persistencia -> segons com es vulgui
No es pregunta al usuari el tipus de persistencia ni els "valors de la BD"
S'ha de deixar la línia de codi segons la persistencia que es vulgui i comentar les altres.
***** En la classe Configurador estan definits els "valors de la BD", si es vol /cal modificar s'ha de fer a "pur codi".*****
2) El métode prinicipal és .ejecutar() se li demana a la Lógica un objecte que implementi la interface Controlador (sense que App sápiga quin és) i se li diu al objecte que implementa la interface Vista que "treballi" amb aquest objecte.
La Lógica sap en tot moment quin controlador ha de proporcionar, explicat més endavant.

3) Segons el controlador, la vista solicita / mostra dades al usuari i solicita proporciona dades al controlador.

PAQUET MODELS

1) Arbre, Flor i Decoració només es diferencien en que tenen un atribut propi. Per a fer més senxill el codi i la(es) BD s'ha obtat per crear només una classe ProductoCompleto que representi a tots (gestionant els camps null).

2) Degut a que un producte pot tenir N unitats en stock i n1, n2, n3, etc unitats en diferents tickets s'ha && en previsió del disseny de la BBDD seguint les normes de ORM && per a facilitar la serialització-> es crea la classe ProductoUnidad, per a guardar la informació de les unitats de un determinat ProductoCompleto en un determinat "llistat de productes".

3) No hi ha cap diferencia substancial entre un ticket i un stock. Els 2 tenen un id, un valor total, i tenen un llistat de ProductesUnidad (producte + unitats d'aquest). La única diferencia és que en un stock només poden haver-hi productes d'una determinada categoria.
S'obta per crear la classe ConjuntoProductos que representi als dos. Per a gestionar la "peculiaritat" del stock s'usa el id.

> id dels stocks [1,2,3] és equivalent a les posicions en el enum Categoria
> id > Categoria.length -> és un ticket
> **Si en un futur variés el número de categories dels productes (varia el num. de stocks) s'hauria de gestionar / actualitzar els id dels tickets existents**

4) Classe Estado i Manager explicats més endavant.

PAQUET UTILS

Classes reutilitzables per a altres projectes.

PAQUET PERISTENCIA + PERSISTENCIAXXXs
Interfaces del patrón DAO + implementacions per a cada familia de persistencia.
GenericDao s'ha creat per a seguir el patró, però la App no requereix métodes que siguin comuns a tots els models.
Es vol capturar les excepcions en les Vistes, per a mostrar el missatge d'error en cas que es produeixin. **De moment només llancen IOException.**
Per a la persistencia en MySQL i Mongo caldrà fer modificacions en Conexion.
En les implementacions dels Daos per a txt:

> Els métodes es podrien simplificar, però estan implementats així per a poder reutilitzar la classe JsonManager.
> Tenen atributs (i imports) per a poder saber els noms dels camps a buscar. **Augmenta la complexitat i el número d'acoplaments, però permet modificar el nom dels atributs dels model sense que calgui canviar el codi.**

PAQUET CONTROLADORS + IMPLEMENTACIONS (LOCALS)

Controladors principals (interfaces + implementacions): 10
8 opcions en el menú (sortir no té cap controlador) + funiconalitat inicial + menú:

Start, RealizarVenta, ShowStocks, NuevoProducto, ModificarUnidades, Descatalogar, ShowValor, ShowFacturación, ShowTickets, Menú

Per reduïr duplicitat de codi es creen interfaces auxiliars + aquestes tindrán implementacions en classes abstractes + les classes que requereixin d'aquests métodes heredaran d'aquestes:

Persistencia (controladores que accedan a la persistencia).
ElegirProducto (controladores que requieran buscar un producto en stock).
GestionarUnidades (controladores que impliquen modificar unidades en stock).
ShowConjuntos (controladores que tenen que mostrar stocks o tickets).

Les implementacions dels controladors principals tenen que implementar la interface Controlador (única classe que coneix la App, necessari per a poder aplicar técnica doble despacho usant la classe ControladorVisitor, explicat més endavant).

ModificarUnidades i ShowTickets no té cap métode propi, però és necessari crear una interface particular per a poder emmagatzemar i distingir el controlador d'aquesta funcionalitat. Les implementacions sobreescriuen algun métode heredat o implementen algun métode abstracte.

Les implementacions dels controladors principals tenen que acavar heredant de LocalControladorManager. Aquesta classe té un atribut de tipus Manager (en models). El Manager té informació dels models que no varia en el temps (i que proporciona als controladors) + té el Estado (Enum) de la aplicació.
El Estado permet saber a la Lógica "quin controlador té que proporcionar a la App". Quan un controlador finalitza la seba funcionalitat canvia el valor del Estado.

La implementació de la Logica crea el Manager + inicialitza els controladors (proporcionant el manager) + sap, per a cada Estat possible, quin controlador té que proporcionar a la App.

No em convenç que els controladors existeixin "sempre", voldria reduïr el "temps de vida d'aquests", però de moment només se m'acut solucionant-ho amb un switch (massa llarg =smell code).

Revisar:

tots els métodes amb parametres tenen assert?
ordre dels métodes máximitzen comprensió del codi?
noms i ingles vs espanyol ok?

PAQUET VISTAS + VISTAS CONSOLA + VISTAS CONSOLA MODELOS

ConsolaV és la vista que implementa Vista (interface amb la que treballa la App).
ConsolaV reb un controlador que implementa Controlador (interface amb la que treballa la App). Però no sap quin tipus de controlador és. Per evitar males práctiques (ex: instanceOf) s'aplica la técnica del doble despacho (el patró visit aplica aquesta tècnica, però són coses diferents):

1) Se li diu al controlador que executi el métode .accept(this)
    Métode que implementen totes les classes que implementen la interface Controlador
2) El método .accept requereix un ControladorVisitor com argument, per tant Vista hereda d'aquesta interface i ConsolaV l'implementa.
3) En totes les implementacions del método accept(ControladorVisitor controlador) el codi és igual, encara que signifiquen coses diferents: {controlador.visit(this)}
    El controlador li diu al ControladorVisitor que executi el mètode .visit, enviant-se ell mateix com a paràmetre.
4) ConsolaV, al implementar ControladorVisitor, té sobrecàrrega de mètodes .visit. Un per a cada controlador principal. El this només pot ser una implementació de un controlador principal -> s'executa el mètode visit corresponent a aquest tipus de controlador.

Cada mètode .visit instancia una vista (segons el cas) i li passa el controlador.
ConsolaV acava sent com "un repartidor de joc": distingeix el controlador i delega la funcionalitat a una altra vista proporcionant-li un controlador d'algun tipus dels principals.
En general hi ha una vista per a cada controlador. Hi han dues parelles de controladors (showValor-showFacturación i showStocks-showTickets) que comparteixen la mateixa vista (ShowTotalesV i ShowConjuntosV).

En el cas que hi hagués interfaces gràfiques, aquestes vistes podrien heredar / compartir classes que implementessin "la part gràfica de la app".

ElegirProductoV es una vista abstracta de la qual hereden les vistes de les funcionalitats que requereixen que el usuari seleccioni un producte del stock (ModificarUnidadesV, DescatalogarProductoV, RealizarVentaV).

Mètode updateProduct / finalizar no em convenç degut a codi duplicat, es pot millorar.
Usa parcialment la vista ShowConjuntosV, solicitant previament al controlador de la funcionalitat un controlador ShowStocksC.

Totes les vistes, excepte MenuV, s'apoien en la vista auxiliar EndV (métodes comuns / similars en les vistes de funcionalitats que implicaran accedir a la persistencia).

Quan una vista ha de mostrar / solicitar dades d'algun model, es delega la tasca a la classe vista encarregada de mostrar / solicitar dades d'aquell model (paquet vistas.consola.modelos).