

Pràctica 3: mercat ub

5 de maig de 2021

noah márquez vara & guillem navarra panos

GRUP F

**Pràctica 3**

**Classes implementades**

1. **Classe Article**

Emmagatzema tota la informació rellevant dels articles. Conté un constructor que permet inicialitzar els valors dels atributs de la classe:

* ***String id***, identificador de l’article (codi alfanumèric).
* ***String nom***, cadena de text que descriu el seu nom.
* ***float preu***, preu unitari de l’article.
* ***int temps***, temps fins l’enviament expressat en minuts.
* ***boolean admetUrgent***, l’article pot permetre o no l’enviament urgent.

També estan especificats els corresponents *getters & setters* dels atributs anteriors.

Finalment conté un mètode ***toString*** que permet imprimir la informació de l’article.

1. **Classe Client**

És una classe abstracta la qual guarda tota la informació dels clients. Aquesta informació inclou:

* ***String email***, correu electrònic.
* ***String nom***, nom del client.
* ***String adreca***, adreça postal del client.

Incorpora un constructor que inicialitza els atributs esmentats anteriorment, així com els *getters & setters* necessaris.

A més, tenim declarats tres mètodes abstracters:

* ***public abstract String tipusClient()***, retorna el tipus de client.
* ***public abstract float calcMensual()***, calcula la quota mensual de cada client depenent del tipus.
* ***public abstract float descompteEnv()***, descompte en tant per cent que s’aplica a les despeses d’enviament.

També tenim definit un mètode ***toString*** que crea una cadena de text amb la informació de cada client.

1. **Classe ClientEstandard**

Subclasse de la classe ***Client***. Conté els mateixos atributs que la seva classe mare i implementa els mètodes esmentats anteriorment. El client estàndard no té descompte en les despeses d’enviament, és per això que conté un atribut de tipus ***public static final int DESCOMPTE = 0***.

El mètode ***public String tipusClient()*** retorna “Estàndard”.

El mètode ***public float calcMensual()*** retorna 0 ja que el client de tipus estàndard no paga cap quota mensual.

El mètode ***public float descompteEnv()*** retorna la costant **DESCOMPTE** mencionada anteriorment.

1. **Classe ClientPremium**

Subclasse de la classe **Client**. Conté els mateixos atributs que la seva classe mare i implementa els mètodes esmentats anteriorment. El client premium té un descompte del 20% en les despeses d’enviament, és per això que conté un atribut de tipus ***public static final int DESCOMPTE = 20***.

El mètode ***public String tipusClient()*** retorna “Premium”.

El mètode ***public float calcMensual()*** retorna 4, ja que la quota del client premium és de 4€ mensuals.

El mètode ***public float descompteEnv()*** retorna la costant **DESCOMPTE** mencionada anteriorment.

1. **Classe Comanda**

S’utilitza per gestionar les comandes. Per tal de vincular un article amb un client, la classe ***Comanda*** conté com atributs un objecte de cada tipus (***Client & Article***). També té un atribut que indica la quantitat d’articles del mateix tipus que s’inclouran en la comanda (***int quantitat***). Consta també d’un atribut de tipus ***Date data***, que ens permet emmagatzemar la data de creació de la comanda.

La classe ***Comanda*** es defineix com una classe abstracta que incorpora un constructor per paràmetres que inicialitza els atributs esmentats anteriorment i els *getters & setters* necessaris. Els mètodes de la classe són els següents:

* ***public float calcPreu()***, calcula el preu de la comanda sense incloure les despeses d’enviament.
* ***public abstract String tipusComanda()***, indica el tipus de la comanda (Normal o Urgent).
* ***public abstract boolean comandaEnviada()***, indica si la comanda ha estat enviada.
* ***public abstract boolean comandaRebuda()***, indica si la comanda ha estat rebuda.
* ***public abstract float preuEnviament()***, indica l’import de les despeses d’enviament.
* ***Public String toString()***, permet obtenir la informació associada a una comanda.

1. **Classe ComandaNormal**

Subclasse de la classe ***Comanda***. Conté els mateixos atributs que la seva classe mare i implementa els mètodes esmentats anteriorment. La comanda normal conté dues constants:

* ***public static final float PREU\_ENVIAMENT = 1***, indica el preu d’enviament d’una comanda normal.
* ***public static final int TEMPS\_REBUDA = 2880***, temps que es triga a rebre una comanda normal desprès de ser enviada (en minuts), és a dir, 2 dies.

El mètode ***public String tipusComanda()*** retorna “Normal”.

El mètode ***public boolean comandaEnviada()*** fa la diferència entre la data actual i la data de creació de la comanda i ho passa a minuts, per després comprovar si aquesta diferència és més grans al temps d’enviament (expressat en minuts) associat a l’article afegit a la comanda. Si la diferència és més gran, retorna ***true*** sinó ***false***.

El mètode ***public boolean comandaRebuda()*** primerament comprova si la comanda ha estat enviada o no, perquè si la comanda no ha estat enviada, tampoc pot ser rebuda, així ens estalviem càlculs. Si per contra la comanda ha estat enviada, farem la diferència entre la data actual i la data de creació de la comanda i ho passem a minuts, per després comprovar si aquesta diferència és més gran al temps d’enviament (expressat en minuts) associat a l’article afegit a la comanda més la constant ***TEMPS\_REBUDA***(indicada anteriorment). Si la diferència és més gran, retorna ***true***, sinó ***false***.

El mètode ***public float preuEnviament()***, retorna la constant ***PREU\_ENVIAMENT*** (indicada anteriorment).

1. **Classe ComandaUrgent**

Subclasse de la classe ***Comanda***. Conté els mateixos atributs que la seva classe mare i implementa els mètodes esmentats anteriorment. La comanda urgent conté dues constants:

* ***public static final float PREU\_ENVIAMENT = 4***, indica el preu d’enviament d’una comanda normal.
* ***public static final int TEMPS\_REBUDA = 1440***, temps que es triga a rebre una comanda normal desprès de ser enviada (en minuts), és a dir, 1 dia.

El mètode ***public String tipusComanda()*** retorna “Urgent”.

El mètode ***public boolean comandaEnviada()*** fa la diferència entre la data actual i la data de creació de la comanda i ho passa a minuts, per després comprovar si aquesta diferència és més grans al temps d’enviament/2 (perquè la comanda urgent triga la meitat del temps a ser enviada que una comanda normal; expressat en minuts) associat a l’article afegit a la comanda. Si la diferència és més gran, retorna ***true*** sinó ***false***.

El mètode ***public boolean comandaRebuda()*** primerament comprova si la comanda ha estat enviada o no, perquè si la comanda no ha estat enviada, tampoc pot ser rebuda, així ens estalviem càlculs. Si per contra la comanda ha estat enviada, farem la diferència entre la data actual i la data de creació de la comanda i ho passem a minuts, per després comprovar si aquesta diferència és més gran al temps d’enviament/2 (expressat en minuts) associat a l’article afegit a la comanda més la constant ***TEMPS\_REBUDA***(indicada anteriorment). Si la diferència és més gran, retorna ***true***, sinó ***false***.

El mètode ***public float preuEnviament()***, retorna la constant ***PREU\_ENVIAMENT*** (indicada anteriorment).

1. **Classe Llista**

El nostre programa ha de gestionar tres tipus d’objectes: ***Article***, ***Client*** i ***Comanda***. Atès que les operacions que necessitem dur a terme són molt similars per les tres col·leccions, hem implementat aquestes operacions en una classe genèrica: ***Llista***. A partir d’ella derivarem classes per a les tres llistes que necessitem.

Per la implementació de la classe ***Llista*** hem seguit el cos de la implementació que ja ens era donada. Tenim un atribut ***protected ArrayList<T> llista***. En la declaració de l’atribut anterior, ***T*** constitueix un paràmetre de la classe que podrem canviar segons ens convingui. L’ús d’aquest paràmetre incideix en el tipus d’elements que es gestionen en l’atribut ***llista***, de tipus ***ArrayList***. La classe ***Llista*** conté un constructor per defecte que inicialitza l’atribut mencionat anteriorment.

Pel que fa els mètodes, hem implementat els següents:

* ***public int getSize()***, retorna la mida de la llista.
* ***public void afegir(T t)***, mètode per afegir un objecte de tipus genèric ***T*** (***Article, Client, Comanda***) a la llista. Pot llençar diferents tipus d’excepcions: si ja hi ha un client amb el mateix email, si ja hi ha un article amb el mateix id o bé, si s'intenta crear una comanda urgent amb un article que no admet enviament urgent.
* ***public void esborrar(T t)***, mètode per esborrar un objecte de la llista. Pot llençar diferents excepcions: si l’objecte passat per paràmetre no està a la llista o bé si la llista està buida.
* ***public T getAt(int position)***, mètode per obtenir un objecte a una determinada posició de la llista. Pot llençar diferents excepcions: si la llista està buida o bé si la posició no és vàlida.
* ***public void clear()***, mètode per esborrar tota la llista. Si la llista està buida, llença una excepció.
* ***public ArrayList<T> getArrayList()***, retorna un objecte de tipus ***ArrayList<T>*** amb el tipus de llista que estem tractant.

1. **Classe LlistaArticles**

Classe derivada de la classe ***Llista***. Té un constructor per defecte en què es fa un ***super*** perquè la inicialització de la llista ja està especificada a la classe genèrica ***Llista***, per tant no hem de repetir la declaració.

Posteriorment conté dos mètodes:

* ***public void afegir(Article a)***, mètode per afegir un article a la llista. Aquest mètode fa ús d’un mètode privat que explicarem a continuació amb la fi de comprovar si ja hi ha algun article a la llista amb el mateix identificador que el què es vol afegir, si és així, es llençarà una excepció i no s’afegirà a l’article a la llista. Posteriorment fem un ***super.afegir(a)*** perquè la declaració del mètode per afegir un objecte a la llista ja l’hem fet a la classe genèrica.
* ***private void contains(Article a)***, mètode auxiliar que comprova si hi ha algun article a la llista amb el mateix identificador que el què es vol introduir. En cas afirmatiu, llença una excepció.

1. **Classe LlistaClients**

Classe derivada de la classe ***Llista***. Té un constructor per defecte en què es fa un ***super*** perquè la inicialització de la llista ja està especificada a la classe genèrica ***Llista***, per tant no hem de repetir la declaració.

Posteriorment conté dos mètodes:

* ***public void afegir(Client c)***, mètode per afegir un client a la llista. Aquest mètode fa ús d’un mètode privat que explicarem a continuació amb la fi de comprovar si ja hi ha algun client a la llista amb el mateix email que el què es vol afegir, si és així, es llençarà una excepció i no s’afegirà a el client a la llista. Posteriorment fem un ***super.afegir(c)*** perquè la declaració del mètode per afegir un objecte a la llista ja l’hem fet a la classe genèrica.
* ***private void contains(Client c)***, mètode auxiliar que comprova si hi ha algun client amb el mateix email que el què es vol afegir. En cas afirmatiu, llença una excepció.

1. **Classe LlistaComandes**

Classe derivada de la classe ***Llista***. Té un constructor per defecte en què es fa un ***super*** perquè la inicialització de la llista ja està especificada a la classe genèrica ***Llista***, per tant no hem de repetir la declaració.

Posteriorment conté dos mètodes:

* ***public void afegir(Comanda c)***, mètode per afegir una comanda a la llista. Aquest mètode fa ús d’un mètode privat que explicarem a continuació amb la fi de comprovar (en el cas de que sigui una ***ComandaUrgent***) si l’article que es vol associar a la comanda admet enviament urgent, si no és així, es llençarà una excepció i no s’afegirà la comanda a la llista. Posteriorment fem un ***super.afegir(c)*** perquè la declaració del mètode per afegir un objecte a la llista ja l’hem fet a la classe genèrica.
* ***private void comprova(Client c)***, mètode auxiliar per comprovar si s’està intentant crear una comanda urgent amb un article que no admet enviament urgent. Si aquest és el cas, es llença una excepció.

1. **Classe Dades**

És la classe principal del paquet del model, ja que conté i gestiona totes les dades de l’aplicació. Per establir amb claredat tot el que ha de fer aquesta classe, hem seguit la interfície ***InDades***, que proveeix una llista amb els mètodes a implementar.

Conté tres atributs:

* ***private LlistaArticles llistaArticles***, atribut que farem servir per manipular la llista d’articles del nostre mercat.
* ***private LlistaClients llistaClients***, atribut que farem servir per manipular la llista de clients del nostre mercat.
* ***private LlistaComandes llistaComandes***, atribut que farem servir per manipular la llista de comandes del nostre mercat.

Aquests atributs s’inicialitzen en el constructor per defecte de la classe ***Dades***.

A continuació fem una recopilació dels mètodes de la classe ***Dades*** (especificats a partir de la interfície ***InDades***):

* ***afegirArticle***, afegeix un article. Rep per paràmetres el seu identificador, el seu nom, el preu, el temps fins l’enviament i un ***boolean*** indicant si aquest article admet enviament urgent o no.

Es comprova si el preu o el temps són iguals o més petits que 0, en aquest cas es llença una excepció. Si no es dona el cas, es crea un objecte de tipus ***Article*** i es crida al mètode ***afegir*** de l’atribut ***llistaArticles*** per afegir l’article a la llista. Si a l’intentar afegir-ho es detecta que ja hi ha un article amb el mateix identificador, es llença una excepció.

* ***recuperaArticles***, ens serveix per retornar un objecte de tipus ***List<String>*** amb els ***Article*** que tinguem a la llista.
* ***afegirClient***, afeigeix un client. Rep per paràmetres el seu email, el seu nom, la seva adreça postal i un ***boolean*** indicant si aquest client és premium o no. Si és premium es crearà un objecte de tipus ***ClientPremium***, sinó, es crearà un objecte de tipus ***ClientEstandard***. En tots dos cassos, es crea l’objecte i s’afegeix a la llista mitjançant el mètode ***afegir*** de l’atribut ***llistaClients***. Si a l’intentar afegir-lo es detecta que ja hi ha un client amb el mateix email, es llença una excepció.
* ***recuperaClients***, ens serveix per retornar un objecte de tipus ***List<String>*** amb els ***Client*** que tinguem a la llista.
* ***afegirComanda***, afegeix una comanda. Rep per paràmetres la posició de l’article a associar a la comanda, la posició del client que fa la comanda, la quantitat d’articles i un ***boolean*** indicant si la comanda és urgent o no. Primer de tot es comprova si alguna de les dues llistes (d’articles o de clients) és buida, ja que si alguna d’elles (o totes dues) estan buides, no es podrà crear una comanda i es llençarà una excepció. Si la comanda és premium, es crearà un objecte de tipus ***ComandaUrgent***, sinó, es crearà un objecte de tipus ***ComandaNormal***. En tots dos cassos, es crea l’objecte i s’afegeix a la llista mitjançant el mètode ***afegir*** de l’atribut ***llistaComandes***

Si la comanda és urgent, i es detecta que l’article que es vol associar a aquesta no admet enviament urgent, es llença una excepció.

Si es detecta que o bé la posició indicada de l’article o del client és errònia (més gran que la mida de la llista, negativa...) es llença una excepció.

* ***esborrarComanda***, fem servir aquest mètode per eliminar una comanda a una determinada posició de la llista especificada per l’usuari. Si la posició introduïda és negativa o bé excedeix la mida actual de la llista, es llença una excepció. Si pel contrari la posició és correcta, però la comanda associada a aquesta posició ja ha estat enviada, es llença una excepció. Sinó, es procedeix a esborrar la comanda a la posició indicada.
* ***recuperaComandes***, ens serveix per retornar un objecte de tipus ***List<String>*** amb les ***Comanda*** que tinguem a la llista.
* ***recuperaComandesUrgents***, ens serveix per retornar un objecte de tipus ***List<String>*** amb les ***ComandaUrgent*** que tinguem a la llista.
* ***static Dades load***, permet carregar les dades del mercat des d’un fitxer. Rep per paràmetres la ruta del fitxer des d’on es vol carregar les dades. Pot llençar vàries excepcions (ex: si la ruta al fitxer és incorrecte, si no existeix cap fitxer en la ruta, si hi ha un problema de lectura, si no es pot tancar el fitxer...).
* ***guardaDades***, mètode que permet guardar un arxiu amb les dades del mercat pel seu posterior ús. Rep per paràmetres la ruta al fitxer on volem guardar les dades. Pot llençar vàries excepcions (ex: problema d’escriptura, no es pot tancar el fitxer...).

Els mètodes proveïts per ***Dades*** seran utilitzats per la classe ***Controlador***. Perquè les dades del model puguin carregar-se i guardar-se correctament, totes les classes involucrades han d’implementar la interfície ***Serializable***.

1. **Classe Controlador**

Serà utilitzada per intervenir entre la vista i les dades del model. Amb aquest objectiu, la classe ***Controlador*** farà servir una instància de la classe ***Dades***, per ser la classe principal del nostre model.

Farà de pont entre la vista i el model. La vista només podrà utilitzar aquesta classe per accedir a la informació del model.

Conté un atribut (com hem mencionat anteriorment) de tipus ***Dades***, anomenat ***dades***. Aquest s’inicialitza en el constructor per defecte de la classe ***Controlador***.

Els mètodes d’aquesta classe criden als mètodes de la classe ***Dades*** ja esmentats anteriorment. En el cas dels mètodes per recuperar els clients, els articles i les comandes, es realitza un tractament especial, si obtenen una llista buida, retornen un ***String*** dient que la llista està buida, sinó, retornen la llista en qüestió.

1. **Classe MercatUB**

Defineix el menú d’opcions i la lògica de l’aplicació. Conté un atribut privat de tipus ***Controlador***. Conté també un mètode públic anomenat ***gestioMercatUB()*** (es crida des de la classe ***IniciadorMercatUB***), on s’implementa el menú de l’aplicació.

1. **Classe IniciadorMercatUB**

Té com a responsabilitat llançar el bucle principal del nostre programa. La classe de la vista **IniciadorMercatUB** té un mètode estàtic **main()** on es crea un objecte de tipus **MercatUB** anomenat mercatUB. Després crida el mètode **gestioMercatUB()** de l’objecte mercatUB, on es troba el bucle principal de l’aplicació.

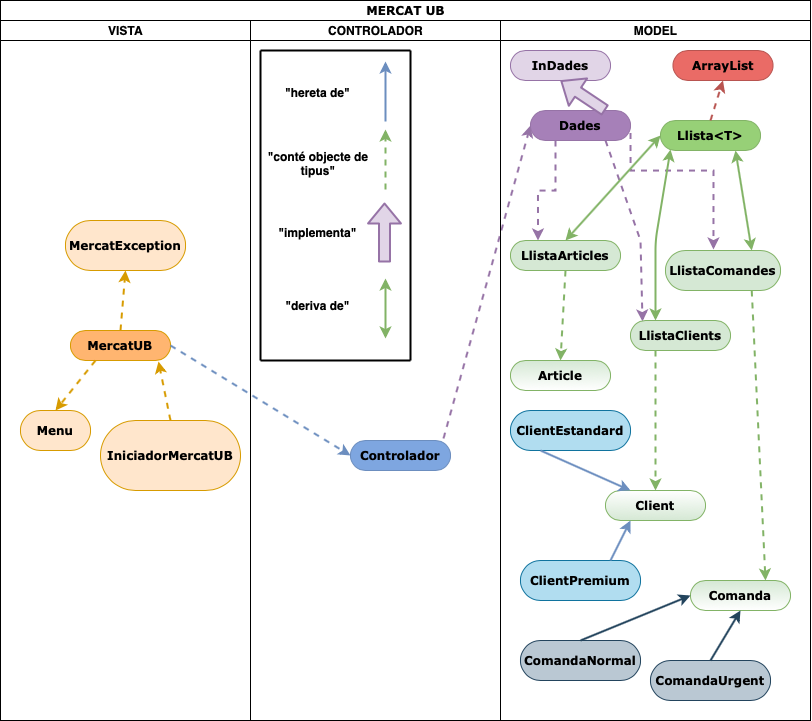
1. **Classe Menu**

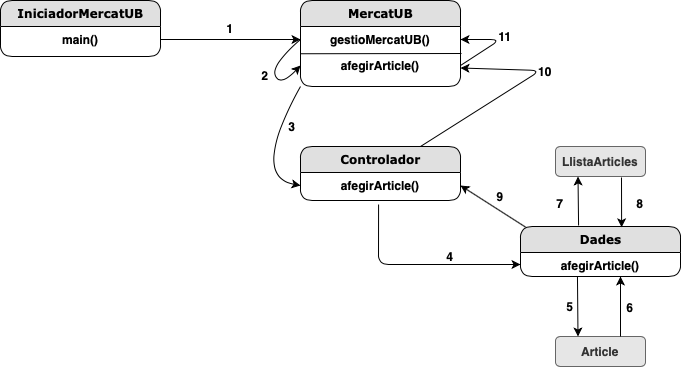
Classe encarregada del funcionament i gestió del menú i submenús d’opcions del programa. Comprova que l’input de l’usuari sigui correcte (que seleccioni una de les opcions del menú).

1. **Classe MercatException**

Per tal de fer la gestió d’errors, com per exemple, intentar afegir un article amb un identificador i que ja existeixi un article amb el mateix identificador a la llista. La classe **MercatException** està implementada dins el paquet de la vista per tal de representar tots els errors particulars.

La classe fa un extends de Exception, té un constructor sense paràmetres que realitza un super i un constructor per paràmetres que rep el missatge de l’excepció produïda.

**Diagrama de relacions entre classes**

**Diagrama de flux quan s’afegeix un article**

**Atributs de la classe Dades**

* ***private LlistaArticles llistaArticles***; ja que el mètode **recuperaArticles** necesita una llista d’articles per a buscar l’article demanat i fer una llista amb el toString de l’article indicat. El mateix passa amb **afegirArticle**.
* ***private LlistaClients llistaClients***; ja que el mètode **recuperaClients** necesita una llista de clients per a buscar el client demanat i fer una llista amb el toString del client indicat. El mateix passa amb **afegirClient**.
* ***private LlistaComandes llistaComandes***; ja que el mètode **esborrarComanda** necesita la llista de comandes de la qual es vol esborra una comanda en particular. El mateix passa amb **afegirComanda**, **recuperaComandes** i **recuperaComandesUrgents**.

**Classe Dades d’un company: modificacions en la vista i el controlador i com es relaciona la interfície InDades amb la pregunta**

Degut que la implementació de la classe ***Dades***es fa seguint la interfície ***InDades***, no s’hauria de canviar res ja que tot i que la implementació pot ser diferent, els noms dels mètodes i de la seva funció ha de ser la mateixa. En resum, si s’ha seguit la interfície ***InDades***, no s’hauria de fer modificacions ni a la vista ni en el controlador (sempre que es mantinguin els noms dels mètodes i la seva funcionalitat).

**Implementació de LlistaArticles, LlistaClients i LlistaComandes de forma lliure però sense classes genèriques: implementació, avantatges i inconvenients**

Probablement, el que fariem seria implementar una classe per a cada una d’aquestes llistes de forma independent, és a dir, sense cap classe mare que les unís ja que en dificultaria l’accés si no es poden fer servir *templates*. Trobo que això potser aportaria una mica de facilitat a l’hora de escriure el codi, però seria un impediment a l’hora de fer-ne manteniment o de llegir-lo o ampliar-lo.

**Proves realitzades per comprovar el correcte funcionament de la pràctica**

Per comprovar el correcte funcionament de la pràctica, a banda de comprovar la funcionalitat de les 5 opcions del menú, s’ha comprovat sobretot que funcionés la persistència de dades, és a dir, la possibilitat de guardar la informació del mercat i recuperar-la posteriorment. A continuació una llista de les comprovacions realitzades per veure el correcte funcionament d’aquesta pràctica:

1. Intentar afegir una comanda sense tenir cap article o client a les llistes, per comprovar com se’ns avisa que no es pot afegir una comanda, ja que no tenim com a mínim un article i un client a qui associar-la.
2. Comprovació de la possibilitat d’afegir un article i posteriorment visualitzar-lo. Comprovar que si s’introdueix un preu de l’article o un temps d’enviament igual a zero o negatiu, es llenci una excepció.
3. Intentar afegir un nou article amb un identificador d’un article que ja es troba a la llista, per comprovar el funcionament de les excepcions.
4. Tornar a intentar afegir una comanda, per veure com se’ns avisa de que no tenim cap client a la llista, per tant no podem afegir una comanda.
5. Comprovació de la possibilitat d’afegir un client i posteriorment visualitzar-lo.
6. Intentar afegir un nou client amb un email d’un client que ja es troba a la llista, per comprovar el correcte funcionament de les excepcions.
7. Comprovació de la possibilitat d’afegir una comanda (ara ja si, degut a que ja tenim com a mínim un article i un client a qui associar-la) i posteriorment visualitzar-la. Crearem una comanda urgent i una de normal, per comprovar que l’opció de només visualitzar les comandes urgent funciona correctament.
8. Intentar associar un article que no admet enviament urgent a una comanda de tipus urgent, per comprovar el funcionament de les excepcions.
9. Comprovació de la possibilitat d’esborrar una comanda donada una posició. Comprovar que si s’introdueix una posició errònia, salta una excepció.
10. Intentar associar un client/article amb una posició errònia a l’hora de crear una comanda, per comprovar el correcte funcionament de les excepcions.
11. Guardar les dades del mercat al disc. S’ha comprovat que funcionés correctament i que es creés el fitxer de manera correcta en el disc. Aquesta opció pot llençar vàries excepcions, ja sigui per un problema d’escriptura o perquè no es pot tancar el fitxer, per exemple.
12. I per últim, recuperar les dades del mercat guardada en el disc. S’ha comprovat que fos capaç de llegir fitxers des del disc. Aquesta opció també pot llençar vàries excepcions: si no es troba el fitxer, si hi ha un problema de lectura, si no es pot fer *càsting* de l’objecte del fitxer a un de tipus ***Dades***, entre d’altres.

**Observacions generals**

Amb aquesta pràctica hem vist i posat a prova diferents eines del llenguatge de programació Java que ens permeten fer herència entre classes, gestionar excepcions, utilitzar classes genèriques i derivades i dur a terme la persistència de les dades, entre d’altres. Tot això ens ha servit per gestionar diferents situacions i controlar diferents tipus de dades fins arribar a desenvolupar un sistema de gestió de comandes per a una companyia que treballa en l’àmbit del comerç electrònic. Ens ha servit per a poder familiaritzar-nos amb eines més complexes i versàtils i poder aprendre dels nostres errors i fer un aprenentatge prova-error, així, els temes explicats a classe queden molt més clars.