Pràctica 3

1 Descripció del Problema

Se'ns ha demanat desenvolupar un sistema de gestió de comandes per a una companyia que treballa en l'àmbit del comerç electrònic. Els principals elements que la companyia fa servir són els articles que ven, els clients i les propis comandes.

Articles. Els articles tenen un identificador únic que consisteix en un codi alfanumèric, una cadena de text que descriu el seu nom i un preu unitari. Els articles poden admetre o no enviament urgent, i quan són enviats, l'enviament es fa utilitzant uns braços robòtics que col·loquen l'article en un embalatge per a la posterior distribució per part de l'empresa de transports. A causa de la completa automatització de sistema, cada article té assignat un temps fins a l'enviament expressat en minuts.

Clients. Cada client ve identificat de manera unívoca per la seva adreça de correu electrònic. Addicionalment, també es requereix saber el seu nom i la seva adreça de correu postal per poder enviar les comandes. Hi ha dos tipus de clients, el client estàndard i el client premium. Els clients estàndard no paguen cap mensualitat, mentre que els clients premium paguen una mensualitat de 4 euros al mes. D'altra banda, els clients premium reben un descompte del 20% en les despeses d'enviament.

Comandes. Un client pot fer una comanda d'una determinada quantitat d'articles d'un mateix tipus, és a dir, que una mateixa comanda no pot contenir articles diferents (fem això per simplificar la pràctica). Les comandes emmagatzemen informació sobre el client i l'article involucrats, així com la quantitat d'unitats de l'article sol·licitades. D'altra banda, necessitem també conèixer la data en la qual la comanda va ser realitzada per, d'aquesta manera, poder fer un seguiment de la mateixa. Hi ha dos tipus de comandes, la comanda normal i la comanda urgent. Cada comanda té un temps fins l'enviament donat per la data de creació de la mateixa i el temps d'enviament de l'article. La comanda urgent triga la meitat d'aquest temps a enviar-se. D'altra banda, un cop enviat, una comanda té un temps fins a la recepció. Es triga 2 dies a rebre la comanda normal i 1 dia la comanda urgent. Quan es mostra la informació relacionada amb una comanda, és molt important determinar si la comanda està enviat o no, i si la comanda ha estat rebuda o no. Es pot consultar la

Figura 1 per tenir una visió general del procés. Les comandes es poden cancel·lar sempre que no hagin estat enviades. D'altra banda, les comandes tenen un preu corresponent a l'enviament i un altre dels articles que incorpora. La comanda normal té un cost d'enviament d'1 euro, mentre que l'urgent costa 4 euros. El preu dels articles es calcula multiplicant la quantitat d'aquests pel cost unitari de l'article associat a la comanda.

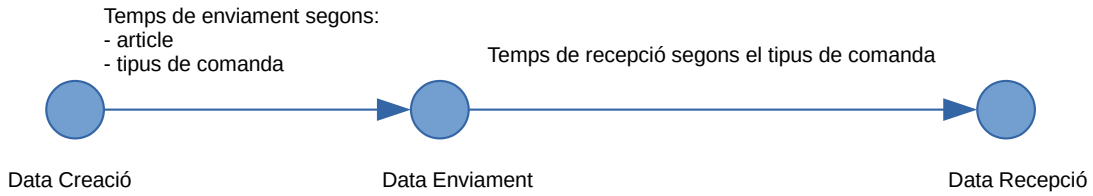

Figura 1: Evolució temporal d'una comanda.

2 Objectiu i Funcionalitats

Es pretén crear un sistema d'informació que modeli un mercat de comerç electrònic on es recullin les especificacions que s'han detallat en la secció anterior. Amb aquest objectiu, igual que en les pràctiques anteriors, seguirem un paradigma de programació orientada a objectes.

L'aplicació a desenvolupar ha d'oferir les següents funcionalitats:

1 Gestió Artícles: Dóna pas a un menú que permet gestionar la informació relacionada amb els articles.

1.1 Afegir Article: Afegeix un nou article al sistema. Entre altres dades, l'aplicació ha de sol·licitar que s'introdueixi el temps en minuts fins a l'enviament de l'article així com si aquest permet enviament urgent o no.

1.2 Visualitzar Articles: Mostra tots els articles afegits.

1.3 Sortir: Torna al menú principal.

2 Gestió Clients: Mostra un menú que permet gestionar els clients.

2.1 Afegir Client: Incorpora un nou client a l'aplicació. L'aplicació ha de preguntar si el client a afegir és estàndard o premium.

2.2 Visualitzar Clients: Visualitza tots els clients afegits.

2.3 Sortir: Torna al menú principal.

3 Gestió Comandes: Visualitza un menú per gestionar les comandes.

3.1 Afegir Comanda: Crea una nova comanda. L'aplicació ha de sol·licitar la informació necessària per vincular un article amb un client, establir la quantitat d'unitats de l'article que es sol·liciten i determinar si la comanda és urgent o no.

3.2 Esborrar Comanda: Esborra una comanda determinada, sempre que no hagi estat enviada encara.

3.3 Visualitzar Comandes: Visualitza totes les comandes realitzades.

3.4 Visualitzar Comandes Urgents: Visualitza únicament les comandes urgents.

3.5 Sortir: Torna al menú principal.

4 Guardar Dades: Guarda les dades de l'aplicació.

5 Carrega Dades: Carrega les dades de l'aplicació.

6 Sortir: Surt de l'aplicació.

3 Material per a la Pràctica

Per a aquesta pràctica es proporcionen les següents classes i interfícies:

- **Menu**: classe útil per implementar els menús textuals descrits anteriorment.
- **InDades**: interfície que ha de ser implementada per la classe Dades.
- **Llista**: classe genèrica per implementar llistes que necessitarem en el nostre sistema d'informació. El fitxer conté l'estructura general de la classe, incloent atributs però només

els prototips dels mètodes, quedant pendent la definició del cos dels mateixos.

4 Descripció de la Pràctica

A continuació es detallen una sèrie de passos per a resoldre la pràctica que s'ha proposat. Es recomana seguir aquests passos.

4.1 Creació del projecte

El primer pas serà crear un projecte NetBeans, el qual s'ha de nomenar com Cognom1Nom1Cognom2Nom2, on 1 i 2 fan referència als dos membres de la parella, tenint en compte les consideracions següents:

- La primera lletra de cada part en majúscula i la resta en minúscula.
- S'han d'evitar els accents i caràcters especials com ñ o ç.

  Per exemple, una estudiant amb nom Dolça Martínez Castanya, hauria de crear un projecte amb el nom MartinezCastanaDolca.
- La classe principal s'ha de anomenar **IniciadorMercatUB**, i el paquet per defecte ha de ser **prog2.vista**. En el nom del paquet s'han d'utilitzar també els mateixos criteris anteriors.
- En NetBeans s'ha d'indicar la classe principal com **prog2.vista.IniciadorMercatUB**.

4.2 Paquets del Projecte

El projecte que hem de desenvolupar està basat en el patró Model-Vista-Controlador. Haurem de crear una classe controlador, de manera que la vista no accedirà directament a les dades del model, sinó que ho farà a través d'aquesta classe. En conseqüència, el projecte estarà format per 3 paquets:

- **prog2.vista**: la vista contindrà totes les classes relacionades amb el maneig del menú d'opcions. Entre elles destaca la classe **MercatUB,** que implementarà el menú d'opcions ofert per l'aplicació.
- **prog2.controlador**: el paquet controlador únicament contindrà la classe **Controlador**, que farà de pont entre la vista i el model. La vista només podrà utilitzar aquesta classe per accedir a la informació del model.

- **prog2.model**: el model contindrà totes les classes que modelen les dades que s'han de gestionar dins de l'aplicació. Dins el model, mereix especial menció la classe **Dades**, que contindrà totes les dades de l'aplicació i portarà a terme totes les accions que afecten les mateixes.

Figura 2: Classes principals del patró Model-Vista-Controlador i les seves relacions.

A la Figura 2 es mostren les relacions entre les classes principals de cada un dels paquets.

4.3 Classes del Projecte

A continuació es descriuen les principals classes que s'han d'implementar dins de la nostra aplicació.

Classe Article

La classe **Article** emmagatzema tota la informació rellevant dels articles. S'ha d'afegir un constructor que permeti inicialitzar els valors dels atributs de la classe (per a més informació sobre aquests atributs, es pot consultar la descripció de el problema a l'inici d'aquest document). També s'han d'afegir els corresponents setters i getters.

S'ha de crear també un mètode **toString** que permeti imprimir la informació de l'article. La cadena de text generada hauria de ser la següent:

```
Id=XXXX, Nom=Artículo A, Preu=100.0, Temps fins enviament=5, Enviament Urgent=false
```

Classe Client y Classes Filles

La classe **Client** guardarà la informació dels clients. Aquesta informació inclou el correu electrònic, nom i adreça de cadascun. S'ha de declarar la classe **Client** com a classe abstracta, incorporant un constructor que inicialitzi els seus atributs, així com els getters i setters necessaris. A més, caldrà que declarem els següents mètodes abstractes, que retornen el tipus de client, calculen la quota mensual de cada client depenent del tipus, així com el descompte en tant per cent que s'aplica a les despeses d'enviament:

```
public abstract String tipusClient();
public abstract float calcMensual();
public abstract float descompteEnv();
```

A més, també necessitem definir un mètode **toString** que ha de crear una cadena de text com la següent:

```
Tipus=Premium, Email=client_a@gmail.com, Nom=Client A, Adreça=Plaça Universitat,
Descompte Enviament=20.0, Mensualitat=4.0.
```

Un cop creada la classe **Client**, procedim a definir dues classes filles, **ClientEstandard** i **ClientPremium**, que implementaran els mètodes abstractes esmentats més amunt.

Classe Comanda y Classes Filles

La classe **Comanda** s'utilitza per gestionar les comandes. Per tal de vincular un article amb un client, es recomana que la classe **Comanda** contingui com a atributs un objecte de cada tipus. També haurà de tenir un atribut que indiqui la quantitat d'articles del mateix tipus que s'inclouran en la comanda i un atribut de tipus **Date**, que permetrà emmagatzemar la data de creació de la comanda. Per poder declarar objectes de la classe **Date**, serà necessari importar **java.util.Date**. Quan es crea un objecte de la classe **Date** amb **new**, dit objecte emmagatzemarà la data actual. També serà interessant el mètode **getTime**, que permet obtenir la data associada a un objecte de tipus **Date** com el número de milisegons transcorreguts des de l'1 de Gener de 1970 a les 00:00.

La classe **Comanda** es definirà com una classe abstracta que incorporarà un constructor i els getters i setters necessaris. També ha d'afegir un mètode per calcular el preu de la comanda sense incloure les despeses d'enviament:

```
public float calcPreu();
```

A més de l'anterior, la classe **Comanda** incorporarà quatre mètodes abstractes. Aquests mètodes seran útils per saber de quin tipus és la comanda, si ha estat enviada o no, si ha estat rebuda, així com l'import de les despeses d'enviament:

```
public abstract String tipusComanda();
public abstract boolean comandaEnviada();
public abstract boolean comandaRebuda();
public abstract float preuEnviament();
```

També declararem el mètode **toString**, que ens permetrà obtenir la informació associada a una comanda de la següent manera:

```
Tipus=Normal, Article=Article A, Client=Client A, Quantitat=2, Data de creacio=Sat Apr 03 18:54:00 CEST 2021, Enviat=false, Rebuda=false, Preu Articles=200.0, Preu Enviament=0.8
```

Un cop definida la classe **Comanda**, procedirem a crear les classes filles **ComandaNormal** i **ComandaUrgent**, que implementaran apropiadament els mètodes abstractes declarats en la seva superclasse.

Classe Llista y Classes Filles

L'aplicació que hem d'implementar ha de gestionar col·leccions de tres tipus d'objectes: **Article**, **Client** i **Comanda**. Atès que les operacions que necessitem dur a terme són molt similars per a les tres col·leccions, la nostra intenció és implementar la majoria d'aquestes operacions en una classe genèrica, i a partir d'ella derivar classes per a les tres llistes que necessitem.

Les classes genèriques en Java constitueixen una eina molt valuosa per a reutilitzar programari. En concret, són classes similars a les classes convencionals però que reben tipus com a paràmetre a l'hora d'instanciar-se. L'esquelet de la classe genèrica que desenvoluparem en aquesta pràctica ve donat com a material de la pràctica en el fitxer **Llista.java**, i es declara de lasegüent manera:

```
public class Llista<T> implements Serializable {
 protected ArrayList<T> llista;
 …}
```

En la declaració 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**.

A partir de la classe **Llista**, definirem tres classes derivades: **LlistaArticles**, **LlistaClients** i **LlistaComandes**, segons el tipus d'objecte a gestionar. Per exemple, la definició de **LlistaArticles** té la següent forma:

```
public class LlistaArticles extends Llista<Article> implements Serializable {…}
```

A més, la definició de classes derivades ens serà útil per redefinir mètodes de la classe **Llista**, ja que serà necessari gestionar les següents excepcions:

- En **LlistaArticles** no es podran afegir dos articles amb el mateix identificador.

- En **LlistaClients** no es podran afegir dos clients amb el mateix correu electrònic.
- En **LlistaComandes** no es podrà afegir un objecte de tipus **ComandaUrgent** si l'article que ha d'enviar-se no admet enviament urgent.

A fi de detectar si hi ha duplicats a **LlistaArticles** o **LlistaClients**, es recomana implementar un mètode privat **contains** en cada classe que realitzi la comprovació.

Classe Dades

La 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 el que ha de fer aquesta classe, s'ha creat la interfície **InDades**, que proveeix una llista amb els mètodes a implementar.

Els mètodes proveïts per **Dades** seran utilitzats per la classe **Controlador**. No hauria de ser necessari crear mètodes addicionals dins de **Dades** a més del constructor, els mètodes de **InDades** i els dos mètodes relacionats amb la persistència del model que es descriuen a continuació.

Tal com es veia en l'apartat 2, la nostra aplicació ha de ser capaç de guardar i carregar les dades del model a partir de fitxers (persistència de les dades). Es suggereix que aquesta tasca s'implementi a través de dos mètodes, amb el prototip:

```
public void guardaDades(String camiDesti) throws MercatException;
public static Dades carregaDades(String camiOrigen) throws MercatException;
```

Perquè les dades del model puguin carregar-se i guardar-se correctament, recordeu que les classes involucrades han d'implementar la interfície **Serializable**.

Si es necessita informació addicional sobre el tractament de la persistència de les dades en Java, es pot consultar el material de suport per a la pràctica anterior (Pràctica 2) disponible al Campus Virtual.

Classe Controlador

La classe **Controlador**, com ja s'ha explicat anteriorment, 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.

Classe MercatUB

Defineix el menú d'opcions i la lògica de l'aplicació. Definirem una funció anomenada **gestioMercatUB** amb la següent signatura:

```
public void gestioMercatUB();
```

Aquesta funció internament utilitzarà la classe **Menú** que es proporciona com a material de la pràctica.

Classe IniciadorMercatUB

Conté la classe principal de l'aplicació i està ubicada al paquet **prog2.vista**. Declara un objecte de la classe **MercatUB** i crida al mètode **gestioMercatUB** esmentat anteriorment.

Classe MercatException

S'ha de crear la classe **MercatException** dins el paquet **prog2.vista**. Aquesta classe serà utilitzada per al maneig de les excepcions, tal com s'explica més avall.

5 Excepcions

Igual que en la pràctica anterior, la gestió dels errors es durà a terme per mitjà d'excepcions. Un exemple d'una operació que pot causar una excepció és quan intentem afegir un nou article amb el mateix codi que un altre prèviament afegit, tal com es deia en l'apartat anterior.

El primer pas per implementar la gestió d'excepcions és crear la classe **MercatException** anteriorment esmentada. Un cop creada (i tal com vam veure en la pràctica anterior), hem de fer servir **throw** des dels mètodes en què es pugui produir algun error. Farem servir la paraula clau **throws** a la fi de la capçalera del mètode en cas que vulguem delegar la captura d'una excepció fins al mètode on es vagi a gestionar. El constructor de l'excepció ha d'utilitzar-se per introduir el missatge descrivint cada tipus d'error. Finalment, utilitzarem un bloc **try-catch** per capturar l'excepció i informar l'usuari de l'error produït i controlar el flux d'execució.

Recordeu que es pot consultar informació addicional sobre la gestió d'excepcions en el material de suport per a la pràctica anterior (Pràctica 2) disponible al Campus Virtual.

6 Format del Lliurament

El lliurament consistirà en tot el codi generat a partir dels diferents punts de l'enunciat, juntament amb la documentació especificada en aquest apartat.

En concret, s'ha de generar un fitxer comprimit (ZIP) amb el nom dels membres de la parella: **Cognom1Nom1Cognom2Nom2_L3**, que contingui:

A El projecte complet de NetBeans.

Tot el codi generat ha d'estar correctament comentat per poder executar el javadoc, generant automàticament la documentació en línia del codi.

B La memòria del lliurament.

La memòria ha de contenir els apartats descrits en la normativa de pràctiques i els punts següents:

1. Explicar les classes implementades.
2. Dibuixar el diagrama de relacions entre les classes que has utilitzat en la teva pràctica. No cal incloure la llista d'atributs i mètodes.
3. Feu un diagrama de flux per mostrar el recorregut que fa el vostre programa quan s'executa l'opció d'afegir un article al sistema. Especificar els mètodes que es criden en cadascuna de les classes. Feu servir fletxes i númerosper indicar l'ordre de les crides.
4. Explicar quins són els atributs de la classe **Dades** i per què.
5. Imagina que un company teu et passés la seva implementació de la classe **Dades** per tal de substituir la del teu codi, quines modificacions serien necessàries en la vista i el controlador? Com es relaciona la interfície **InDades** amb la pregunta anterior? Justifica la resposta.
6. Si tinguessis llibertat per implementar **LlistaArticles**, **LlistaClients** i **LlistaComandes** com volguessis, però no poguessis fer servir classes genèriques (com **Llista <T>**), com portaries a terme la implementació? Analitza les avantatges i inconvenients de la solució adoptada.
7. Detallar les proves realitzades per comprovar el correcte funcionament del codi, resultats obtinguts i accions derivades.
8. Observacions generals.

7 Data límit del Lliurament

Consultar el calendari de lliuraments al Campus Virtual.