PRÀCTICA 4 MERCATUB

MercatUB

Oscar de Caralt Roy MC

Pau Segura Baños MC

ENGINYERIA INFORMÀTICA UB

Primer de carrera: Programació 2

21/05/2021

ÍNDEX:

Introducció:

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 pròpies comandes.

Per a realitzar això es requereixen uns coneixements bàsics i un cert domini de POE (programació orientada a esdeveniments), herència de classes, de polimorfisme, de creació de classes i interficies i d'encapsulament.
Aquesta pràctica era diferent a totes les anteriors ja que, a part de fer que el codi funcionés correctament, també ens havíem de preocupar de l'apartat gràfic i del disseny de les interfícies.

Hem fet una breu explicació de cada classe, però l'explicació de cada mètode està inclosa als respectius javadocs.

1- Expliqueu quines classes has pogut reutilitzar de la pràctica 3 per a fer aquest. Quins canvis sobre les classes reutilitzades has necessitat fer i perquè.

Hem pogut reutilitzar totes les classes del *package* model i el *package* controlador:

➔ **Articles:** aquesta classe conté un constructor que inicialitza l'objecte article amb els atributs codi, el nom del producte,el preu, si admet un enviament urgent o no, i el temps. Cada atribut té el seu respectiu getter i setter, i la clase també conté un mètode toString().

- **Client:** és una classe abstracta que conté un constructor que inicialitza l'objecte client amb els atributs nom, correu electrònic i la direcció. Cada atribut té el seu respectiu getter i setter, i la clase també conté un mètode toString().

- **ClientEstandard:** aquesta classe hereda de la classe client per crear un client estandard. Té els mateixos atributs, però la característica més ressenyable que té és que no paga cap mensualitat ni té cap descompte.

- **ClientPremium:** aquesta classe hereda de la classe client per crear un client premium. Té els mateixos atributs, però la característica més ressenyable que té és que paga mensualitat de 4 euros i té un descompte de 20%.

- **Comanda:** aquesta classe abstracta crea una comanda amb diferents articles per cada client. El constructor inicialitza l'obkecte comanda amb els atributs client, articles, quants articles te demanats i la data de creació de la comanda. La data i la quantitat d'objectes tenen getter i setter. Declara 4 mètodes abstractes: tipusComanda, comandaEnviada, comandaRebuda i preuEnviament. Implementa dos mètodes: calcPreu, per calcular el preu de la comanda sense l'enviament i el toString.

- **ComandaNormal:** hereda la classe comanda i implementa els mètodes declarats a la classe comanda.

- **ComandaUrgent:** hereda la classe comanda i implementa els mètodes declarats a la classe comanda.

- **InDades:** declara els mètodes que s'han d'implementar a la classse Dades.

- **Dades:** Serà la classe principal del package model, on implementarem tots els mètodes que posteriorment seran cridats al menú pel controlador, és a dir, és la columna vertebral del nostre treball ja que és on podrem ajuntar la resta de classes implementades anteriorment i donar forma al nostre projecte.

➔ **Llista:** aquesta és una classe genèrica que declara els mètodes per treballar amb arraylist. Es declaren d'una manera genèrica(amb T) per no haver-los de declarar tres vegades, per cada tipus d'objecte que hi ha. Es declaren els mètodes, getSize,afegir(T t), esborrar(T t), getAt(position), clear, isEmpty i getArrayList.

➔ **LlistaArticles:** Aquesta classe també hereda de la classe Llista. S'han d'especificar els mètodes que necessiten un objecte i implementar els mètodes amb objectes de la classe Articles en comptes d'objectes de la classe genèrica

➔ **LlistaClients**: Aquesta classe també hereda de la classe Llista. S'han d'especificar els mètodes que necessiten un objecte i implementar els mètodes amb objectes de la classe Clients en comptes d'objectes de la classe genèrica

➔ **LlistaComandes:** Aquesta classe també hereda de la classe Llista. S'han d'especificar els mètodes que necessiten un objecte i implementar els mètodes amb objectes de la classe Comandes en comptes d'objectes de la classe genèrica.

Però com que la vista només es pot relacionar amb el controlador hem hagut de modificar o afegir alguns mètodes del controlador.
Els canvis principals realitzats han estat els següents:

- Hem modificat els tres mètodes per afegir, perquè abans teníem *systems outs* que, en la interfície, no havien de sortir.

- Hem afegit tres mètodes per recuperar les List<String> de clients, articles i comandes, que aquests mètodes s'utilitzen per mostrar les llistes en la interfície.

- Hem hagut de posar entre comentaris les excepcions que es llençaven al controlador per evitar que sortissin per consola.

Hem modifcat el package vista en la seva totalitat .

AppMercatUB: És la classe principal, des d'on es llença el programa.
Hem d'incloure un objecte de la classe controlador per accedir als mètodes del package controlador.
Al disseny d'AppMercatUB tenim 5 botons que ens permeten moure'ns per la nostra app. Ens deixen: Gestionar els articles, Gestionar els clients, Gestionar les comandes, gravar les dades i recuperar-les.
En la source tenim un atribut de tipus controlador i els constructors / mètodes / events de la classe AppMercatUB explicats cadascun en els javadocs.

FrmGestioArticle: És un JFrame connectat a l'App. Quan a l'App presiones el botó gestió d'articles s'obre aquesta pantalla.
El controlador es passa per paràmetre pel constructor.
Al disseny de FrmGestioArticle tenim 1 jList que ens permet mostrar els articles que hi han actualment i 2 botons que ens permeten moure'ns per la nostra app. Ens deixen: afegir un article o tornar a la pantalla principal.
En la source tenim un atribut de tipus controlador i els constructors / mètodes / events de la classe FrmGestioArticle explicats cadascun en els javadocs.

FrmGestioClients:És un JFrame connectat a l'App. Quan a l'App presiones el botó Gestio Clients s'obre aquest frame.
El controlador es passa per paràmetre pel constructor.
Al disseny de FrmGestioClients tenim 1 jList que ens permet mostrar els clients que hi han actualment i 2 botons que ens permeten moure'ns per la nostra app. Ens deixen: afegir un client o tornar a la pantalla principal.
En la source tenim un atribut de tipus controlador i els constructors / mètodes / events de la classe FrmGestioClients explicats cadascun en els javadocs.

FrmGestioComandes: És un JFrame connectat a l'App. Quan a l'App presiones el botó Gestió Comandes s'obre aquest frame.

El controlador es passa per paràmetre pel constructor.
Al disseny de FrmGestioComandes tenim 1 jList que ens permet mostrar les comandes que hi han actualment i 3 botons que ens permeten moure'ns per la nostra app. Ens deixen: afegir una comanda, esborrar una comanda (sempre i quan aquesta no s'hagi enviat encara) o tornar a la pantalla principal.
En la source tenim un atribut de tipus controlador i els constructors / mètodes / events de la classe FrmGestioComandes explicats cadascun en els javadocs.

FrmAfegirClient: És un JDialog connectat al FrmGestioClients. Quan al Frm pitjes el botó Afegir Client salta aquesta JDialog.
Aquesta classe te configurades les excepcions, perque saltin amb JOPtionPane, enlloc que surtin per consola.
Al disseny de FrmAfegirClient tenim 2 botons (acceptar i cancelar l'operació), 1 JCheckBox per a saber si és premium o no i 4 jTextFields que l'usuari omplirà amb el que es demana en les seves corresponents etiquetes.
En la source tenim un atribut de tipus controlador i els constructors / mètodes / events de la classe FrmAfegirClient explicats cadascun en els javadocs.
El constructor d'aquesta classe rep un objecte de la classe controlador, un boolean i un Java.awt.

FrmAfegirArticle:És un JDialog connectat al FrmGestioArticle. Quan al Frm pitjes el botó Afegir Article salta aquesta JDialog.
Aquesta classe te configurades les excepcions, perque saltin amb JOPtionPane, enlloc que surtin per consola.
Al disseny de FrmAfegirClient tenim 4 JTextField per posar el codi, el nom de l'article, el preu i els minuts que trigarà en arribar. Hi ha un JCheckBox per a dir si l'article admet enviament urgent o no, i dos botons, per acceptar o cancelar la operació.
En la source tenim un atribut de tipus controlador i els constructors / mètodes / events de la classe FrmAfegirArticle explicats cadascun en els javadocs.
El constructor d'aquesta classe rep un objecte de la classe controlador, un boolean i un Java.awt.

FrmAfegirComandes: És un JDIalog connectat al FrmGestioComandes. Quan al Frm pitjes el botó Afegir Article salta aquesta JDialog.
Aquesta classe té configurades les excepcions, perque saltin amb JOPtionPane, enlloc que surtin per consola.
El constructor d'aquesta classe rep un objecte de la classe controlador, un boolean i un Java.awt.
Al disseny del FrmAfegirComandes tenim dos JComboBox, cadascun per escollir el client i l'article corresponent per a crear la comanda, un JTextFIeld, per dir quantes unitats tenim i un JCheckBox per marcar si és o no és premium.
A més Tenim dos botons: un per acceptar la comanda, i un per cancelar i no crear-la.
En la source tenim un atribut de tipus controlador i els constructors / mètodes / events de la classe FrmAfegirComandes explicats cadascun en els javadocs.

2- Indiqueu quins tipus d'esdeveniments heu fet servir al vostre codi

La majoria d'esdeveniments que hem utilitzat han sigut esdeveniments d'actionPerformed. A tots els botons hem utilitzat un actionPerformed per designar l'acció que es dóna a terme quan apretem el botó.
Però també hem utilitzat un event d'una llista, el valuechanged, que l'emprem per fer visible un botó. És a dir, quan es selecciona un element de una llista, el botó es torna funcional.
També hem utilitzat JOptionPane per al llançament de les excepcions. És a dir, si s'introdueix quelcom erròniament, salta el JOptionPane.

3- Explica quins canvis hauríeu de realitzar en l'aplicació si es volgués afegir la funcionalitat següent: Al iniciar l'aplicació s'ha d'obrir una subfinestra amb la llista de comandes pendents d'enviar.

Primer hauríem de fer un nou mètode al controlador que comprovi l'estat d'enviament de les comandes, i posés les no enviades en una llista que posteriorment es mostrés
Hauríem de crear un JFrame a la vista, amb una JList on es veiessin les comandes no enviades. Després en AppMercatUB hem de fer un mètode que obri aquest JFrame, i que es cridés al constructor AppMercatUB perquè s'iniciï només posar en marxa l'aplicació.

4- Proves realitzades per comprovar el correcte funcionament de la pràctica, resultats obtinguts i accions derivades

Iniciem el programa i s'obre la pantalla principal amb els cincs botons. A partir d'aquí triem l'opció que volem.

Primer creem un article. Per cada article ens demana el codi, el nom del producte, el preu, si admet o no enviament urgent i el temps d'arribada del producte. Si afegim dos productes amb el mateix codi el programa llança una excepció (JOptionpane) i no afegeix el segon article a la llista.

Un cop hem creat l'article, és el torn de crear un client. Per crear un client ens demana el correu electrònic, el nom, la direcció i si és premium o no. Si hi ha dos clients amb el mateix correu electrònic el programa llança l'excepció (JOptionpane). i no afegeix el segon client a la llista de clients.

Ara creem la comanda, si creem una comanda urgent amb un article que no admet urgent, comprovem que la comanda no s'ha creat i que ha saltat una excepció (JOptionpane).

Ara guardem en un arxiu el que hem fet. Li donem a l'opció de guardar i tanquem l'execució del programa. El tornem a executar i fem l'opció de recuperar l'arxiu. Observem que el programa torni a imprimir la llista de comandes, la d'articles i la de clients i que tot estigui com abans de sortir.

Si tot això s'ha reproduït amb èxit, vol dir que el programa és correcte.

5- Observacions generals.

Aquesta pràctica ens ha resultat més fàcil i entretinguda que la tercera, ja que hem reutilitzat molta part del codi. Només hem tingut problemes en un moment quan passàvem de gestionar clients a gestionar articles ja que se'ns esborraven les dades que creavem. Això era degut a que cada cop que es produïa un canvi de finestra es creava una nova, enlloc de només tancar-la i perdiem les dades presents en l'antiga finestra.

Conclusions:

Hem assolit els objectius personals que ens vam proposar al començament de la pràctica, que eren, anar treballant poc a poc cada dia, però amb una certa constància i tenir el projecte acabat amb 7 dies d'antelació al lliurament, per a si sorgia algun dubte poder preguntar al professor de teòrico-pràctiques Així que els resultats obtinguts en aquest projecte són altament satisfactoris.