PRÀCTICA 2

"CLUB UB"

Oscar Romero Alarcon
Diana Benito Reyes
Programació II
11/04/2021

Índex

1. Introducció i objectius

En aquesta segona pràctica hem dissenyat un programa que gestiona els socis d'un centre excursionista, a través d'un menú de text compost de diverses funcionalitats.

Els objectius de la pràctica són: aprendre el funcionament de les classes abstractes i les interfícies (per exemple, la interfície Serializable), i saber utilitzar-les quan és degut; declarar i utilitzar excepcions; i guardar i carregar dades en un fitxer mitjançant Streams.

2. Preguntes de la pràctica

➔ Explicar les classes implementades.

Per aquesta pràctica hem fet servir els dos paquets vista i model, igual que en la pràctica anterior. Al paquet vista trobem totes les interaccions amb l'usuari que executa el programa; i al paquet model, el desenvolupament intern d'aquestes interaccions.

A més, hem introduït un nou concepte: les interfícies. Es tracten d'una declaració de mètodes buits que posteriorment omplirem i farem servir en una classe determinada.

Classes del paquet vista:

- IniciadorClubUB: en aquesta classe trobem el mètode principal (main) de la pràctica. Aquí (en el main), es crea un objecte de tipus VistaClubUB anomenat "vistaClub" i es crida al mètode gestioClubUB() per inicialitzar l'aplicació.

- VistaClubUB: en aquesta classe es defineixen totes les interaccions del programa amb l'usuari, on es demana tota la informació necessària sobre el club. Es crea el mètode d'objecte gestioClubUB, on s'implementarà el menú de l'aplicació.

- Menu: s'encarrega de mostrar per pantalla les opcions del menú i rep l'opció que l'usuari desitja, comprovant que aquesta és correcta.

- ExcepcioClub: aquesta és la classe que ens permetrà utilitzar excepcions en determinats mètodes de la nostra pràctica. La classe ExcepcioClub hereta de la classe Exception de Java i només té el constructor per defecte i un constructor amb missatge, ja que el seu funcionament està definit en la classe Exception.

Classes del paquet model:

- ClubUB: aquesta classe conté tota la informació sobre el club. S'estableix el preu de les excursions, la quota mensual i el descompte. Es defineixen dos mètodes per tal de guardar i carregar tota la informació que conté aquesta classe al disc de l'ordinador. El mètode que carrega la informació s'ha de definir com un mètode de classe. A més, les classes que tenen objectes que han de ser guardats en un fitxer han d'implementar la interfície Serializable.

- LlistaSocis: aquesta és la classe que agrupa tots els socis en una llista. S'implementen els mètodes declarats en la interfície InSociList, tals com: getSize, addSoci, isFull, etc. La llista es construeix amb un ArrayList, i té una capacitat màxima per defecte de 100 socis; però hem d'introduir l'opció d'introduir la mida per teclat abans de crear la llista. A més cal afegir el mètode contains, que no permet afegir dos socis amb el mateix DNI.

- Soci: és una classe abstracta que serveix per guardar la informació d'un soci. Els atributs principals són nom i DNI. Aquí és on s'implementen els mètodes declarats en la interfície InSoci: els getters i setters per al nom i el DNI, un mètode que calcula el preu de la quota, i un altre que calcula el preu de les excursions. Per últim, ha de tenir un mètode toString() que mostri per pantalla la informació d'un soci.

- SociFederat: hereta de la classe Soci. Té com atribut un objecte de tipus Federacio. Llança una excepció del tipus ExcepcioClub en cas que el preu de la federació sigui menor a 100. S'implementa un mètode toString() que retorna la informació específica d'un soci federat a més de la informació comú en tots els socis.

- Federacio: els socis federats tenen la seva pròpia assegurança de la federació, que inclou un descompte en el preu de les excursions i en la quota mensual del centre. S'implementa un mètode toString() que retorna la informació específica d'un soci estàndard, a més de la informació comuna en tots els socis.

- SociEstandard: hereta de la classe Soci. Conté un objecte de tipus Asseguranca. Llança una excepció quan el tipus d'assegurança no és "Bàsica" ni "Completa". En aquesta classe es defineix un constructor, els getters i setters i un mètode toString().

- Asseguranca: els socis estàndard tenen una assegurança, que pot ser "Bàsica" o "Completa", i un preu. Llavors, en aquesta classe Assegurança, es defineix el mètode constructor, els setters i getters i un toString que mostri la informació per pantalla.

- SociJunior: hereta de la classe Soci. Els socis junior no tenen assegurança i no han de pagar les excursions.

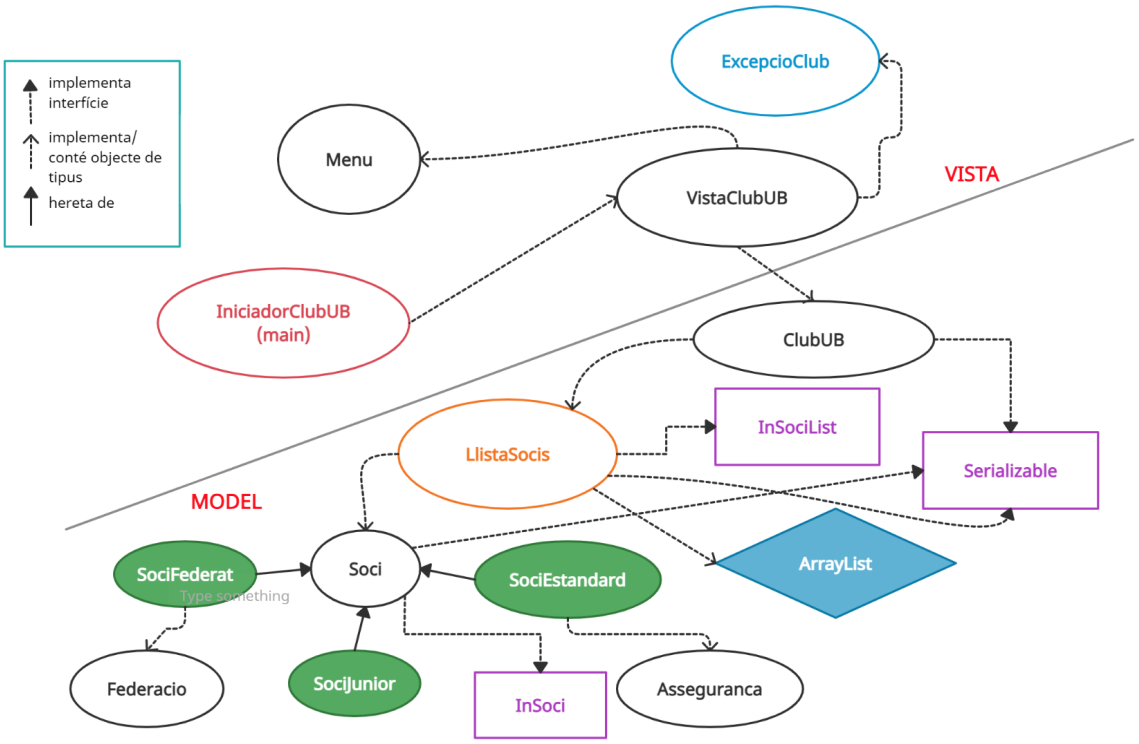
Interfícies (paquet model):

- InSoci: aquesta interfície declara els mètodes necessaris per a la nostra classe Soci.
- InSociList: aquesta interfície declara els mètodes que hem de fer servir a la classe LlistaSocis.

➔ **Explicar com has declarat l'atribut de la classe ClubUB i perquè.**

En la classe ClubUB es declaren quatre variables de tipus float: una per al preu de les excursions, una altra per a la quota mensual, per al descompte de les excursions, i per al descompte de la quota. Altrament, es defineix l'atribut _llistaSocis de tipus LlistaSocis, que farem servir en els mètodes d'aquesta classe per realitzar accions com afegir o eliminar un soci, calcular el preu de la factura mensual d'un soci, etc.

➔ **Dibuixar el diagrama de relacions entre les classes que heu utilitzat a la vostra pràctica. No cal incloure la llista d'atributs i mètodes.**



➔ **Explicar quins són els atributs de la classe Soci i perquè.**

Els atributs de la classe Soci són els dos Strings nom i DNI, ja que són les que caracteritzen i diferencien els socis del club. Implementem en la classe Soci els mètodes getters i setters d'aquests dos atributs, per utilitzar-los posteriorment.

➔ **Per què creieu que la classe Soci ha de ser abstracta?**

Una classe de Java només pot ser abstracta quan és la classe mare en l'herència d'una classe de sèries filles. Com ja sabem, les classes SociFederat, SociJunior i SociEstandard hereten de la classe Soci.

Una classe abstracta implementa almenys un mètode abstracte, és a dir, un mètode sense cos que no pot realitzar cap acció. Aquest mètode serveix per definir quina acció s'ha de fer, però no diu com, ja que deixa que les classes filles ho determinin. En el nostre cas, s'implementa el mètode buit *public abstract float calculaPreuExcursio(float preuExcursioBase)* en la classe Soci, i es defineix en les classes filles, però amb un funcionament diferent per a cadascuna.

➔ **Creieu que podríem haver treballat només amb dues classes Soci i SociFederat? Per què creus que és necessari crear les tres classes Soci, SociEstandard, SociFederat i SociJunior?**

Són necessàries les tres classes filles, ja que aquestes implementen de maneres diferents els mètodes definits en la classe Soci. Veiem un exemple amb el mètode calculaPreuExcursio: per als socis federats, aquest mètode retorna el preu de les excursions amb un descompte; per als socis junior, les excursions són gratuïtes; i per als socis estàndard, s'afegeix al preu base de les excursions una quantitat determinada per un objecte de tipus assegurança. Així, degut a les diferències en la manera de definir els mètodes segons el tipus de soci, comprovem que ens cal tant la classe Soci com les classes filles SociEstandard, SociJunior i SociFederat.

➔ **Indiqueu si hi ha i on es troben els exemples de mètodes polimòrfics al vostre codi.**

En el nostre codi trobem un exemple de polimorfisme al mètode calculaQuota, que està definit en la classe Soci de la següent manera: *public float calculaQuota(float quotaBase) throws ExcepcioClub{ return quotaBase; }*. Per això, es fa servir aquesta implementació en les classes filles SociJunior i SociEstandard, ja que en aquestes dues classes no es torna a definir el mètode calculaQuota. Altrament, en la classe SociFederat trobem la implementació *public float calculaQuota(float quotaBase) throws ExcepcioClub{ return quotaBase * descomptequota/100; }*. D'aquesta manera, tot i que la implementació del mètode calculaQuota per a les classes SociJunior i SociEstandard és la que hi ha definida a la classe Soci, existeix una altra definició per a aquest mètode, que es troba en la classe SociFederat.

➔ **Analitzeu perquè l'objecte de tipus Asseguranca es crea fora del constructor de la classe SociEstandard. Es podria instanciar a dins? Com canviaria el sentit de la implementació?**

Creem l'objecte de tipus Asseguranca en la classe ClubUB i li pasem com a paràmetre a SociEstandard, tot i que també el podriem haver instanciat en el mateix constructor de la classe SociEstandard. Així doncs, hem escollit fer-ho d'aquesta manera per especificar que l'assegurança no depèn del soci, sinó que existeix per

ella mateixa i el soci en fa ús. A més, per als socis estàndard necessitem una assegurança de tipus “Bàsica” o “Completa”, però això no descarta l’opció de que existeixin altres tipus d’assegurança per a altres tipus de socis. En resum, crear l’objecte de tipus Assegurança dins del constructor de SociEstandard limitaria la funcionalitat de la classe Assegurança, ja que aquesta dependria estrictament de la classe SociEstandard.

➔ **Expliqueu com heu utilitzat la classe ExcepcioClub al vostre codi.**

Primer, hem definit la classe ExcepcioClub en la vista del nostre projecte i hem fet que hereti de la classe Excepcion ja definida a Java. Així, la nostra classe només haurà de tenir el constructor per defecte i un constructor amb missatge, ja que el seu funcionament és el de Exception.

Posteriorment, hem afegit “throws ExcepcioClub” en la declaració de tots els mètodes que necessiten fer ús d’una excepció, és a dir, en gairebé totes les classes del paquet model. Per implementar les excepcions en aquests mètodes hem fet servir les funcions *throw* i *try{...} catch{...} finally{...}.*

➔ **Si teniu un soci federat que fa 5 excursions en un mes quina serà la quantitat de la seva factura aquell mes?**

La factura total serà de 97,5 euros.

```
Factura del soci: Oscar amb DNI: 21:
97.5 euros
```

➔ **Si teniu un soci estàndard amb una assegurança de tipus Bàsica d’un preu de 5€ i que fa 5 excursions en un mes quina serà la quantitat de la seva factura aquell mes?**

La factura total serà de 150 euros.

```
Factura del soci: Xavi amb DNI: 34:
150.0 euros
```

➔ **Detallar les proves realitzades per comprovar el correcte funcionament de la pràctica, resultats obtinguts i accions derivades.**

Aquest procediment es pot apreciar en el video de la memòria. Tot i això s’ha tingut en compte tots els possibles casos, casos en que vulguem eliminar socis inexistents, casos en la entrada del tipus d’assegurança sigui incorrecta, casos en que el preu de la federació sigui inferior a 100... i tots estan tractats per les excepcions.

➔ **Observacions generals**

Un dubte que hem tingut ha sigut si al modificar el tipus assegurança caldria també comprovar que aquesta fos Bàsica o General, en el nostre cas ho hem comprovat.

Un altre punt a remarcar, ha sigut el fet de generalitzar les excepcions en una classe ExcepcioClub, d’aquesta manera ha sigut molt més fàcil tractarles i de gran utilitat.

3. Conclusions

Finalment, hem vist que una classe mare (en una herència de classes) és abstracta quan conté un mètode abstracte sense cos, que es defineix en les classes filles. Altrament, una interfície defineix una sèrie de mètodes buits que s'han d'implementar en una classe determinada.

A més, hem après a declarar una classe per les excepcions, a definir-la en els mètodes que la necessiten mitjançant la funció *throws,* i a implementar-la amb l'estructura *try-catch-finally*.

Per guardar dades en un fixter, hem après com utilitzar els Streams d'entrada (InputStream) i de sortida (OutputStream).