CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE – UNIVERSITA' DI BOLOGNA

CORSO DI PROGRAMMAZIONE AD OGGETTI – a. a. 2013/2014

RELAZIONE SULLO SVILUPPO DELL'APPLICAZIONE SIMIL-GESTIONALE

SHOPMANAGE

Lucchi Angela 0000652089

1. ANALISI DEL PROBLEMA

L'applicazione realizzata ha lo scopo di gestire un negozio e il relativo magazzino in maniera semiautomatizzata.

Dal magazzino si può:
-Visualizzare la lista dei prodotti con tutte le loro caratteristiche
-Aggiungere un prodotto
-Controllare se un prodotto è disponibile
-Richiedere altri prodotti al fornitore

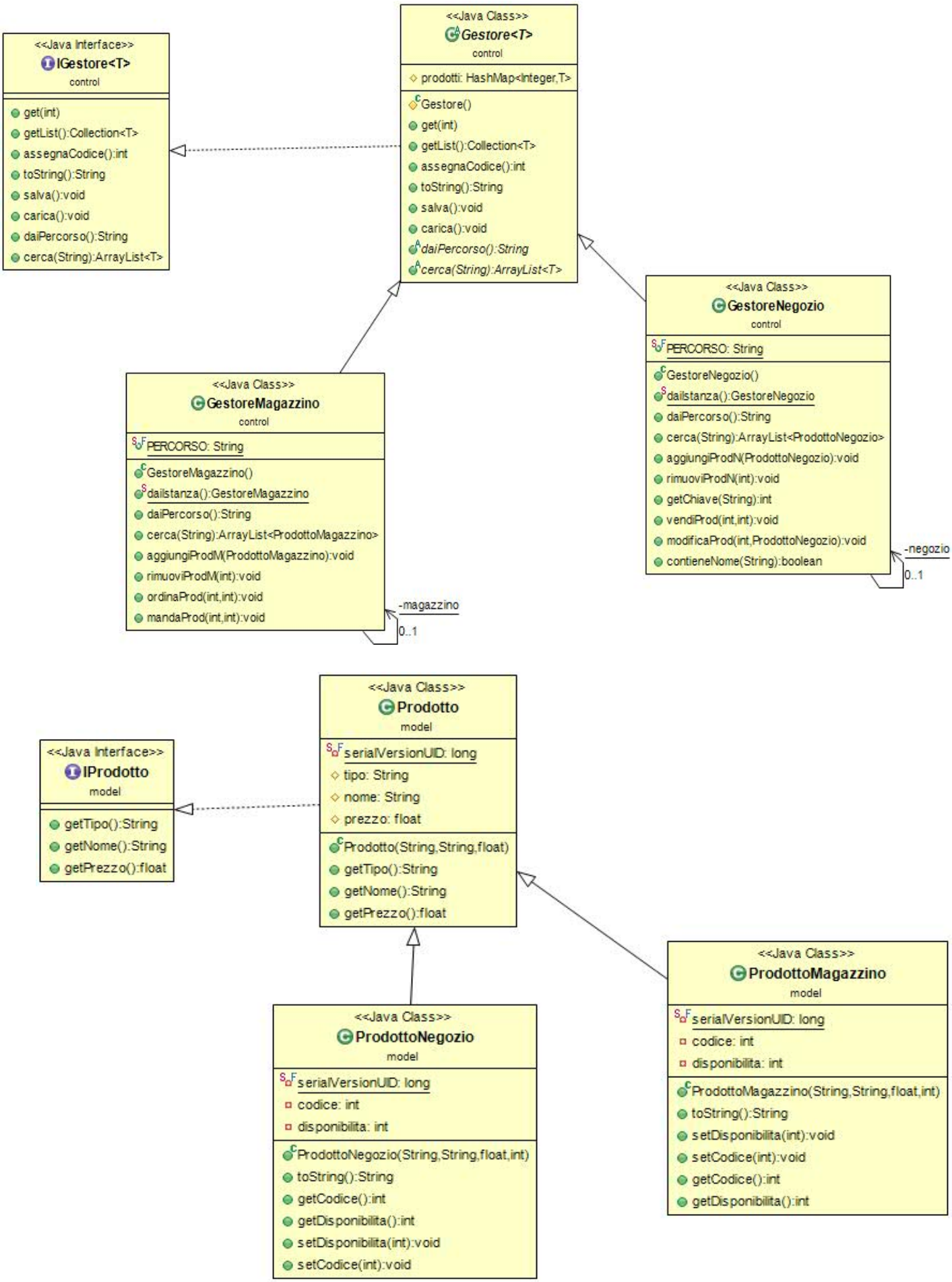
Dal negozio si può:
-Visualizzare la lista dei prodotti con tutte le loro caratteristiche
-Vendere un prodotto
-Controllare se un prodotto è disponibile
-Richiedere altri prodotti al magazzino

In questa applicazione i prodotti una volta ordinati arrivano subito.

Le liste dei prodotti vengono memorizzate in modo persistente infatti ogni volta che un prodotto viene aggiunto, modificato o tolto le liste vengono salvate in automatico dal sistema senza bisogno di interazione da parte dell'utente.

2. PROGETTO ARCHITETTURALE

Principalmente viene usato il pattern singleton, in particolare le classi che ereditano dalla classe Gestore (astratta) istanziano un'unica istanza della classe.
Per semplicità si è deciso di creare un frame per ogni finestra della GUI.


<<Java Interface>>
IGestore<T>
control
get(int)
getList():Collection<T>
assegnaCodice():int
toString():String
salva():void
carica():void
daiPercorso():String
cerca(String):ArrayList<T>
<<Java Class>>
Gestore<T>
control
prodotti: HashMap<Integer,T>
Gestore()
get(int)
getList():Collection<T>
assegnaCodice():int
toString():String
salva():void
carica():void
daiPercorso():String
cerca(String):ArrayList<T>
<<Java Class>>
GestoreNegozio
control
PERCORSO: String
GestoreNegozio()
dailstanza():GestoreNegozio
daiPercorso():String
cerca(String):ArrayList<ProdottoNegozio>
aggiungiProdN(ProdottoNegozio):void
rimuoviProdN(int):void
getChiave(String):int
vendiProd(int,int):void
modificaProd(int,ProdottoNegozio):void
contieneNome(String):boolean
-negozio
0..1
<<Java Class>>
GestoreMagazzino
control
PERCORSO: String
GestoreMagazzino()
dailstanza():GestoreMagazzino
daiPercorso():String
cerca(String):ArrayList<ProdottoMagazzino>
aggiungiProdM(ProdottoMagazzino):void
rimuoviProdM(int):void
ordinaProd(int,int):void
mandaProd(int,int):void
-magazzino
0..1
<<Java Interface>>
IProdotto
model
getTipo():String
getNome():String
getPrezzo():float
<<Java Class>>
Prodotto
model
serialVersionUID: long
tipo: String
nome: String
prezzo: float
Prodotto(String,String,float)
getTipo():String
getNome():String
getPrezzo():float
<<Java Class>>
ProdottoNegozio
model
serialVersionUID: long
codice: int
disponibilita: int
ProdottoNegozio(String,String,float,int)
toString():String
getCodice():int
getDisponibilita():int
setDisponibilita(int):void
setCodice(int):void
<<Java Class>>
ProdottoMagazzino
model
serialVersionUID: long
codice: int
disponibilita: int
ProdottoMagazzino(String,String,float,int)
toString():String
setDisponibilita(int):void
setCodice(int):void
getCodice():int
getDisponibilita():int

3.ORGANIZZAZIONE DEI PACKAGE

I package sono stati suddivisi secondo il pattern MVC (Model – View – Controller).

I package creati sono:

- control: contiene le classi che gestiscono il progetto e implementano i metodi che determinano i cambiamenti nella GUI.
  Ci sono le classi:
    IGestore
    Gestore
    GestoreMagazzino
    GestoreNegozio
- exception: contiene tre classi che estendono exception e permettono di lanciare nuove eccezioni:
    ExceptionEqualName
    ExceptionNegativeNumber
    ExceptionNegativeQty
- model: contiene le classi che modellano gli oggetti:
    IProdotto
    Prodotto
    ProdottoMagazzino
    ProdottoNegozio
- view: contiene la classe che fa partire l'applicazione
    MainFrame
- view.magazzino: contiene le classi che si occupano di visualizzare i frame che sono relativi alla parte del magazzino:
    AggiungiProdotto
    GestioneMagazzino
    MandaFrame
    OrdinaFrame
- view.negozio: contiene le classi che si occupano di visualizzare i frame che sono relativi alla parte del negozio:
    GestioneNegozio
    VendiFrame

4.PROGETTAZIONE DI DETTAGLIO

- Gestore: classe astratta che implementa IGestore e permette di salvare e caricare i dati, gestendoli attraverso una mappa. Il metodo astratto ritorna il percorso del file su cui salvare o da cui caricare i dati. Ci sono infatti 2 diversi file per tener traccia dei prodotti: uno per i prodotti presenti nel magazzino e uno per i prodotti presenti nel negozio.
- Igestore: interfaccia che contiene i metodi per salvare su file, caricare da file e cercare un prodotto.
- GestoreMagazzino: classe con un'unica istanza. Estende Gestore e ne implementa il metodo astratto, fornendo il percorso relativo al file che memorizza i prodotti nel magazzino. Permette di aggiungere, rimuovere, ordinare e mandare prodotti. Ha il metodo cerca per cercare un prodotto in base al nome.
- GestoreNegozio: classe con un'unica istanza. Estende Gestore e ne implementa il metodo astratto, fornendo il percorso relativo al file che memorizza i prodotti nel negozio. Permette di aggiungere e vendere prodotti. Ha il metodo cerca per cercare un prodotto in base al nome.

- ExceptionNegativeQty: eccezione che viene lanciata quando cerco di selezionare una quantità non positiva
- ExceptionNegativeNumber: eccezione che viene lanciata quando cerco di mandare al negozio o di vendere più merce di quella che possiedo.
- ExceptionEqualName: eccezione che viene lanciata quando cerco di aggiungere un prodotto il cui nome esiste già.

- Iprodotto: Interfaccia che contiene i metodi che restituiscono il tipo, il nome e il prezzo del prodotto.
- Prodotto: classe astratta serializzabile che implementa Iprodotto che permette di modellare un prodotto generico e implementa i metodi dell'interfaccia.
- ProdottoMagazzino: classe serializzabile che estende Prodotto, si aggiungono i campi codice e disponibilità e i metodi getter e setter a loro relativi.
- ProdottoNegozio: classe serializzabile che estende Prodotto, si aggiungono i campi codice e disponibilità e i metodi getter e setter a

loro relativi.

- MainFrame: classe che contiene il frame che fa partire l'applicazione. Il bottone Magazzino permette di accedere alla gestione del magazzino, il bottone Negozio permette di accedere alla gestione del negozio, il bottone Chiudi chiude l'applicazione.

- AggiungiProdottoMagazzino: apre il frame per aggiungere un nuovo prodotto, permette di settare tutti i campi (tipo, nome, prezzo, quantità) del prodotto ad eccezione del codice che viene impostato dal sistema. Controlla che il prodotto che si sta inserendo non esista già nel sistema a seconda del nome. Permette anche di tornare indietro alla schermata precedente col tasto "INDIETRO".
- GestioneMagazzino: classe che crea il frame che permette di visualizzare i prodotti presenti in magazzino con tutte le loro caratteristiche: codice, nome, tipo, prezzo e disponibilità. Contiene un JtextField che serve per la ricerca, un bottone cerca che cerca i prodotti il cui nome combacia con la stringa inserita nel JtextField, un bottone "AGGIUNGI" che apre AggiungiProdottoMagazzino e permette di aggiungere un prodotto, un bottone "RIMUOVI" che permette di rimuovere un prodotto dopo averlo selezionato, un bottone "ORDINA"che permette di incrementare la quantità di un prodotto e un bottone "MANDA A NEGOZIO" per mandare un prodotto al negozio. Ha anche il bottone "INDIETRO" per tornare indietro alla schermata precedente.
- MandaFrame: classe che crea il frame che permette di selezionare la quantità di prodotto selezionato da mandare al negozio. Contiene uno spinner che fa selezionare la quantità e un bottone "OK" per confermare. Controlla che la quantità inserita sia positiva e che sia minore o uguale alla quantità posseduta.
- OrdinaFrame: classe che crea il frame che permette di selezionare la quantità di prodotto selezionato da aggiungere a magazzino.Contiene uno spinner che fa selezionare la quantità e un bottone "OK" per confermare. Controlla che la quantità inserita sia positiva.

- GestioneNegozio: classe che crea il frame che permette di visualizzare i prodotti presenti in negozio con tutte le loro caratteristiche: codice, nome, tipo, prezzo e disponibilità. con tutte le loro caratteristiche: codice, nome, tipo, prezzo e disponibilità. Contiene un JtextField che serve per la ricerca, un bottone cerca che

cerca i prodotti il cui nome combacia con la stringa inserita nel JtextField, un bottone “RICHIEDI” per chiedere un prodotto al magazzino aprendo la lista dei prodotti disponibili e in magazzino e usando il bottone “MANDA A NEGOZIO”. Contiene inoltre il bottone “VENDI” che apre VendiFrame per vendere una certa quantità del prodotto selezionato. Ha anche il bottone “INDIETRO” per tornare indietro alla schermata precedente.

- VendiFrame: classe che crea il frame che permette di selezionare la quantità di prodotto selezionato da vendere. .Contiene uno spinner che fa selezionare la quantità e un bottone “OK” per confermare. Controlla che la quantità inserita sia positiva e minore o uguale a quella posseduta.

5.NOTE

Nella classe Gestore, all’interno del metodo carica, è stato inserito un @SuppressWarnings. Questo perché a generarlo è stato il cast, necessario per inserire l’oggetto caricato dalla memoria nella mappa. (this.prodotti = (HashMap<Integer, T>) o) Per risolverlo si sarebbe dovuto inserire un instance of, che però non è utilizzabile in modo generico.