ESAME DI FONDAMENTI DI INFORMATICA T-2 del 9/9/2014
Proff. E. Denti – G. Zannoni
Tempo a disposizione: 4 ore MAX

NB: il candidato troverà nell'archivio ZIP scaricato da Esamix anche il software "Start Kit"

NOME PROGETTO ECLIPSE: CognomeNome-matricola (es. RossiMario-0000123456)

IL consorzio di farmacie *PharmaMe* ha richiesto di sviluppare un'applicazione ad uso delle farmacie aderenti, che permetta la ricerca dei prodotti in base ad una serie di proprietà.

DESCRIZIONE DEL DOMINIO DEL PROBLEMA

Secondo lo schema dell'Agenzia Italiana del Farmaco (AIFA), ogni farmaco è caratterizzato dal suo *Principio Attivo* e da una serie di proprietà, fra cui il *Gruppo di Equivalenza* (si veda oltre), la *Denominazione commerciale* (a volte differenziata per *Confezione*), il *Prezzo al pubblico* (in Euro), la *Ditta Produttrice* e il *Codice AIC* (un numero di 8 cifre che identifica univocamente il farmaco).

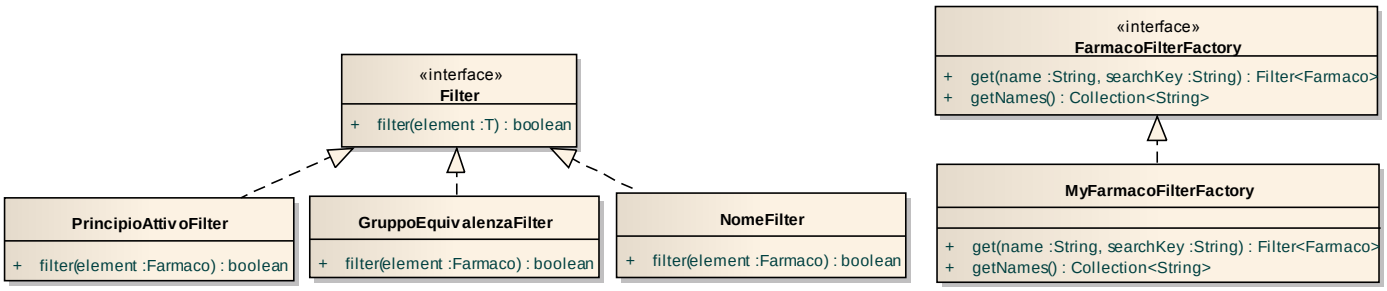
Il *Gruppo di Equivalenza* esprime la tipologia di farmaco indipendentemente dalla sua denominazione commerciale, ma in modo più preciso rispetto al solo principio attivo: ad esempio, al principio attivo "Lansoprazolo" corrispondono ben 73 gruppi di equivalenza, che si differenziano per dosaggio, forma del prodotto, ecc. (uno di essi è "Lansoprazolo 30mg 14 unita' uso orale"). Per comodità, a ogni gruppo di equivalenza è associato un codice alfanumerico univoco di tre caratteri (nel caso precedente, "CDB"), così da identificare rapidamente farmaci equivalenti; tale codice non è, in questo contesto, utilizzato.

Il file di testo Classe_A_per_Principio_Attivo_15.01.2014.csv contiene i dati di tutti i farmaci di classe A, nel formato più oltre specificato.

Parte 1 (punti: 19)

Dati (package pharmame.model) *(punti: 11)*

Il modello dei dati deve essere organizzato secondo il diagramma UML più sotto riportato.



SEMANTICA:

a) la classe ***Farmaco*** (fornita nello start kit) rappresenta un farmaco con tutte le sue proprietà;

b) l'interfaccia ***Filter<T>*** (fornita) dichiara il metodo ***filter*** che esprime l'idea di filtrare un elemento di tipo T;

c) la classe ***NomeFilter*** (da realizzare) implementa direttamente o indirettamente ***Filter<T>*** realizzando il metodo ***filter*** in modo che filtri i soli farmaci la cui denominazione include il nome specificato nel costruttore del filtro;

d) la classe ***PrincipioAttivoFilter*** (da realizzare) implementa direttamente o indirettamente ***Filter<T>*** realizzando il metodo ***filter*** in modo che filtri i soli farmaci la cui descrizione del principio attivo include il principio attivo specificato nel costruttore del filtro;

e) la classe ***GruppoEquivalenzaFilter*** (da realizzare) implementa direttamente o indirettamente ***Filter<T>*** realizzando il metodo ***filter*** in modo che filtri i soli farmaci la cui descrizione del gruppo di equivalenza include la stringa specificata nel costruttore del filtro;

f) la classe di utilità ***FilterApplier*** (fornita) fornisce il metodo statico ***applyFilter*** che rende trasparente l'applicazione di un filtro a una collezione di elementi: invocato su una ***Collection<T>*** e un filtro, restituisce una ***List<T>*** contenente i soli elementi che soddisfano i requisiti del filtro;

g) l'interfaccia ***FarmacoFilterFactory*** (fornita) dichiara due metodi: ***get*** che recupera un ***Filter<Farmaco>*** dati il nome del filtro e la chiave, e ***getNames*** che restituisce tutti i nomi dei filtri disponibili, sotto forma di ***Collection<String>***;

h) la classe ***MyFarmacoFilterFactory*** (da realizzare) concretizza ***FarmacoFilterFactory*** nel caso in questione, accettando le tre stringhe "Nome", "Principio Attivo" e "Gruppo Equivalenza" come identificativi di un'istanza rispettivamente di ***NomeFilter***, ***PrincipioAttivoFilter*** e ***GruppoEquivalenzaFilter***, lanciando ***IllegalArgumentException*** ove venga richiesto un filtro con un nome diverso dai tre leciti sopra specificati.

Lo Start Kit contiene anche i test (da includere nel progetto) per verificare il funzionamento di queste classi.

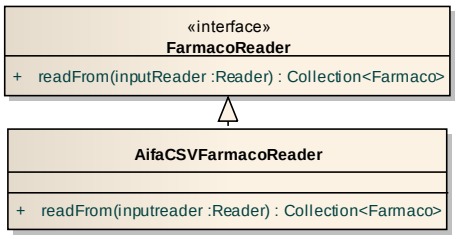
Persistenza (package pharmame.persistence) *(punti 8)*

Come già anticipato, il file di testo Classe_A_per_Principio_Attivo_15.01.2014.csv contiene i dati di tutti i farmaci di classe A, uno per riga: la prima riga, di intestazione, è diversa dalle successive in quanto contiene le descrizioni dei diversi campi possibili (10). Le righe successive descrivono invece i diversi farmaci, specificando una serie di campi separati da "punto e virgola" ('**;**').

Le righe devono contenere sempre almeno sei campi: alcune possono contenere altri campi opzionali, che dovranno essere ignorati. I sei campi fondamentali sono nell'ordine il *Principio Attivo*, il *Gruppo di Equivalenza*, la *Denominazione e Confezione*, il *Prezzo al pubblico*, la *Ditta Produttrice* e il *Codice AIC*. Il terzo di questi (*Denominazione e Confezione*) contiene in realtà due informazioni distinte, separate fra loro da un asterisco ('*'): pertanto, ogni farmaco correttamente descritto è in realtà sempre caratterizzato da 7 proprietà, coerentemente con la lista argomenti attesi dal costruttore di ***Farmaco***.

Se una riga contiene meno di sei campi (separati da ';'), o uno dei campi è la stringa vuota o fatta di soli spazi, o il campo prezzo non è un numero reale, o il campo codice non è un intero, occorre lanciare una opportuna ***BadFileFormatException*** con adeguato messaggio d'errore.

L'architettura software risultante è illustrata nel diagramma UML che segue:



SEMANTICA:

a) l'interfaccia ***FarmacoReader*** (fornita) dichiara il metodo ***readFrom*** che legge una ***Collection<Farmaco>*** dal ***Reader*** ricevuto come argomento, lanciando ***IOException*** o ***BadFileFormatException*** secondo necessità;

b) la classe ***AifaCSVFarmacoReader*** **(da realizzare)** concretizza ***FarmacoReader*** nel caso del file AIFA formattato come sopra specificato.

Lo Start Kit contiene anche i test (da includere nel progetto) per verificare il funzionamento di questa classe.

Parte 2 (punti: 11)

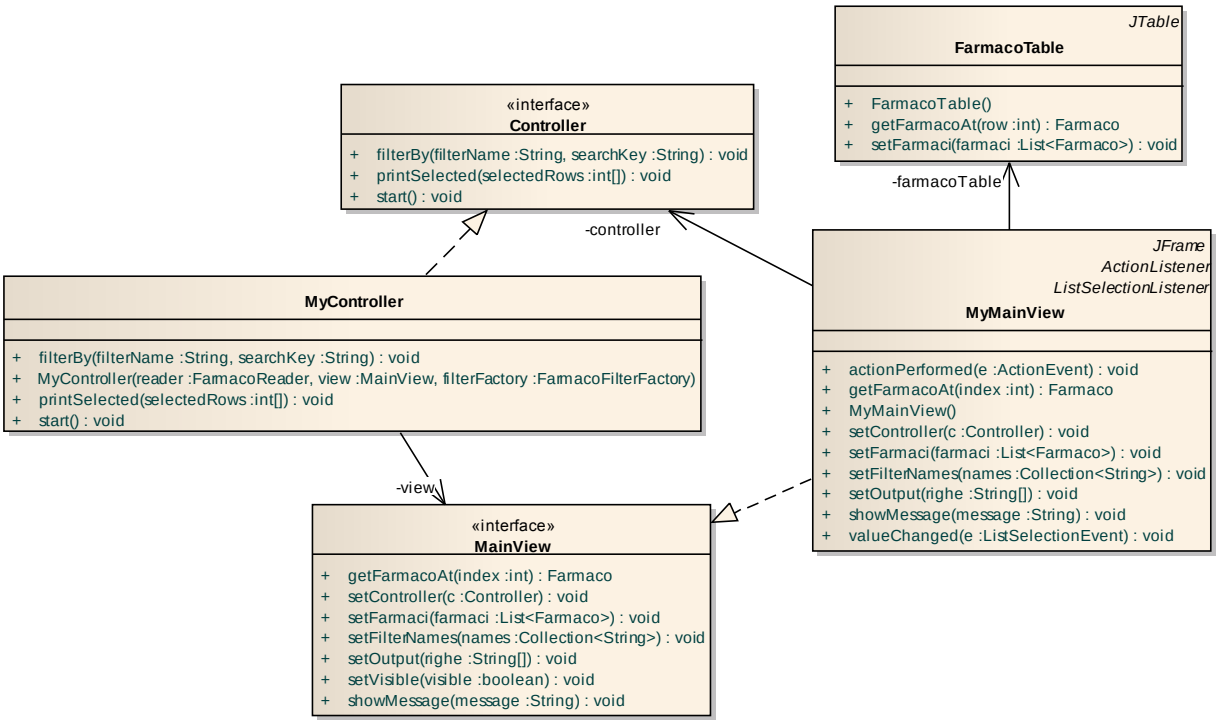
Controller e GUI (package pharmame.ui)

L'interfaccia grafica deve permettere la ricerca dei prodotti per principio attivo, nome medicinale o gruppo di equivalenza, come nell'esempio mostrato di seguito. Inizialmente, la combo in alto contiene i tre filtri possibili e il campo di testo "chiave" è vuoto (Fig. 1). Dopo aver digitato un testo (anche parziale) come chiave e scelto un tipo di filtro, la pressione del pulsante **Filtra** causa il popolamento della sottostante tabella (Fig. 2); in alternativa al pulsante, lo stesso effetto dev'essere ottenibile anche premendo **ENTER** sulla tastiera quando il campo di testo "chiave" è attivo.

Successivamente, il cliente può scegliere una o più righe della tabella: in ogni momento, l'area di testo sottostante mostra i prodotti selezionati *ordinati per prezzo crescente* (Fig. 3, Fig. 4).

La classe ***ProgramStarter*** (fornita nello start kit) contiene il *main* di partenza dell'intera applicazione.

L'architettura software complessiva è illustrata nel diagramma UML che segue:



SEMANTICA:

a) La classe ***FarmacoTable*** (fornita) implementa una versione specializzata di JTable, adatta alla visualizzazione di farmaci. Il solo costruttore presente è quello di default, senza argomenti. Il metodo ***setFarmaci*** imposta la lista di farmaci da mostrare, mentre il metodo ***getFarmacoAt*** restituisce il ***Farmaco*** alla linea i-esima della tabella.
IMPORTANTE: quando si seleziona o deseleziona una riga della tabella viene generato un ***ListSelectionEvent***, da gestirsi a cura di un opportuno ***ListSelectionListener*****.**
SUGGERIMENTO: per evitare doppioni, occorre distinguere gli eventi di "riga selezionata" da quelli di "riga deselezionata"; a tal fine può essere utile il metodo ***getValueIsAdjusting*** di ***ListSelectionEvent***, che è vero solo per gli eventi generati "di rimbalzo" (ossia, quelli di "riga deselezionata").

b) L'interfaccia ***MainView*** (fornita) dichiara sette metodi per agire sulla vista principale dell'applicazione: ***setController*** imposta un ***Controller***, ***showMessage*** visualizza un messaggio, ***setVisibile*** mostra/nasconde la vista, ***setFilterNames*** imposta l'insieme dei nomi dei possibili filtri, ***setFarmaci*** imposta l'insieme la lista dei farmaci disponibili, ***setOutput*** imposta l'insieme delle righe da mostrare nell'area di testo in basso, mentre ***getFarmacoAt*** restituisce il ***Farmaco*** alla linea i-esima della tabella, delegando il lavoro all'omonimo metodo di ***FarmacoTable.***

c) L'interfaccia ***Controller*** (fornita) dichiara tre metodi:

- ***filterBy*** che filtra l'insieme dei farmaci usando come filtro quello passato (per nome) come primo argomento, scegliendo i soli farmaci che contengono la chiave passata come secondo argomento e mostra i farmaci filtrati utilizzando il metodo ***setFarmaci*** della view.
- ***printSelected*** che stampa sull'area di testo (mediante l'uso di ***setOutput*** della view) i (soli) farmaci presenti in tabella alle righe in cui indici sono specificati nell'array di interi ricevuto come argomento, in ordine di prezzo crescente; per recuperare un farmaco dato l'indice, occorre usare il metodo ***getFarmacoAt*** esposto dalla view.
- ***start*** che attiva l'applicazione eseguendo le operazioni che seguono:
  i. apre il file ed effettua la lettura mediante l'apposito reader;
  ii. imposta il controller (se stesso) sulla view;
  iii. imposta i nomi dei filtri sulla view (***setFilterNames***);
  iv. rende visibile la view (***setVisible***).

  Nel caso in cui una delle operazioni di cui sopra sollevi un'eccezione, occorre mostrare a video il messaggio dell'eccezione catturata (***showMessage*** della view).

d) La classe ***MyController*** (da realizzare) implementa ***Controller***: il costruttore riceve il ***FarmacoReader*** da cui leggere i dati, la ***MainView*** da pilotare, e una ***FarmacoFilterFactory*** a cui far costruire i filtri che gli servono.

e) La classe ***MyMainView*** (da realizzare) costituisce la GUI dell'applicazione, operante come sopra descritto.

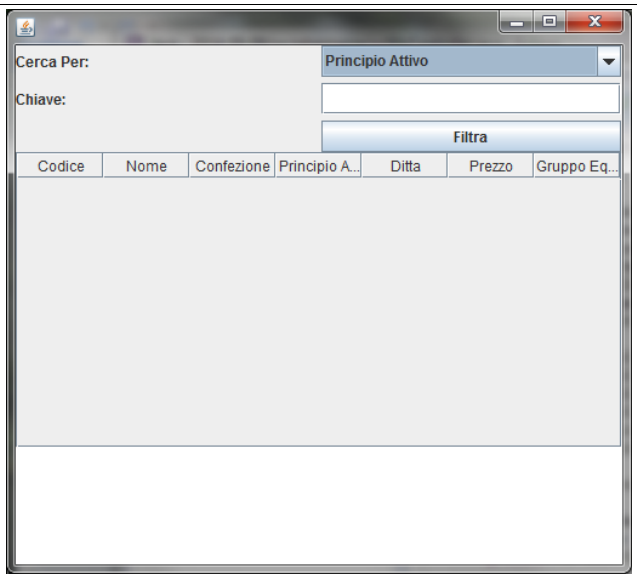


Figura 1

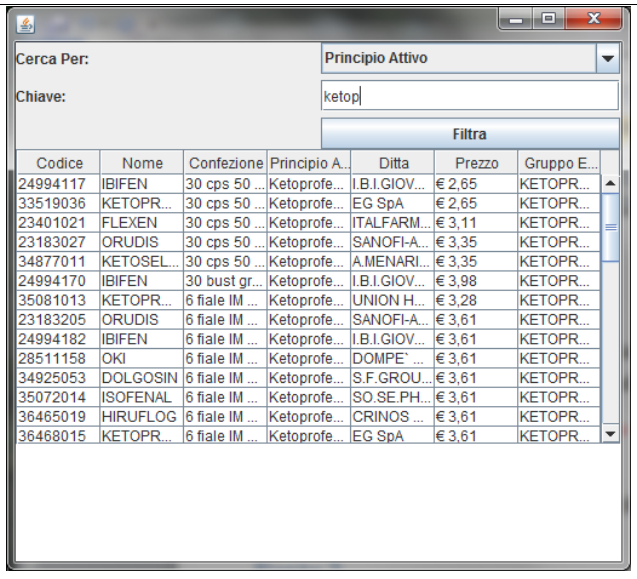


Figura 2

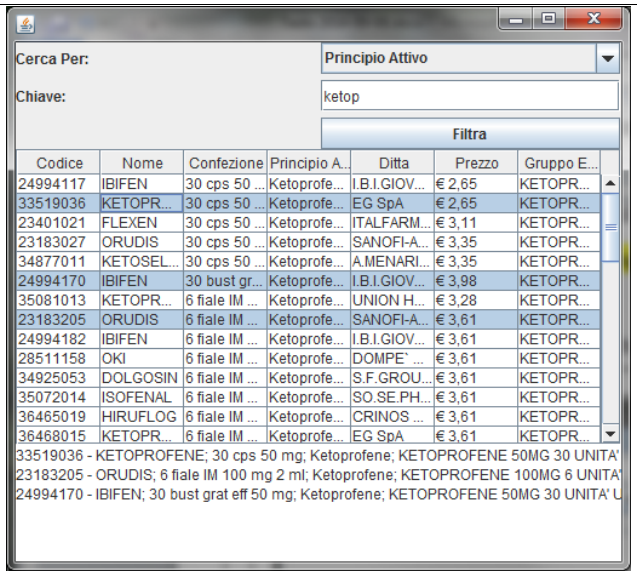


Figura 3

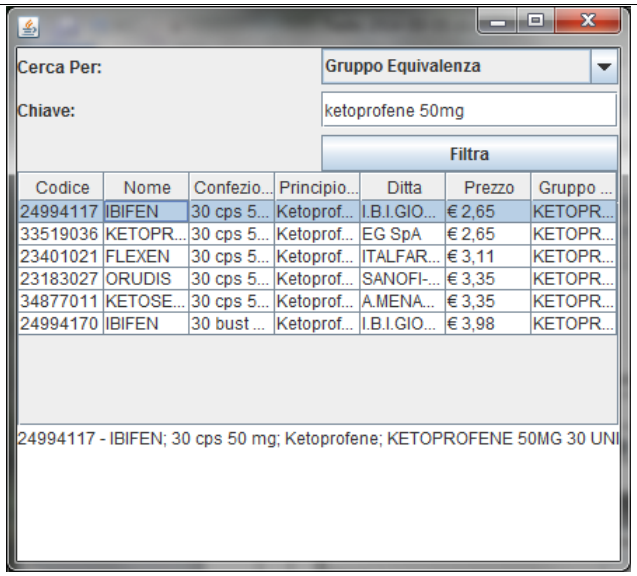


Figura 4