ESAME DI FONDAMENTI DI INFORMATICA T-2 del 16/7/2014
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)

La compagnia *Dent-o-Coffee*, distributore di macchine automatiche per caffè e bevande, ha richiesto di sviluppare il nuovo software di interfaccia, che applichi le proprie innovative politiche commerciali.

DESCRIZIONE DEL DOMINIO DEL PROBLEMA

Nei distributori automatici tradizionali, il denaro da inserire dipende soltanto dalla bevanda scelta: ogni bevanda ha un prezzo fisso, inserito il quale la macchina eroga le bevanda.

Nei distributori automatici di Dent-o-Coffee, invece, ***il prezzo della bevanda non è fisso, ma può variare fra un MINIMO e un MASSIMO secondo una molteplicità di fattori***. Più precisamente, a partire dal ***prezzo BASE*** di quella bevanda, si applicano ***sconti*** o ***incrementi di costo***, in base:

- *allo specifico cliente*, riconoscibile tramite ID inserito prima della scelta della bevanda;
- *agli specifici consumi del cliente*, valutati secondo opportune ***strategie di sconto*** (configurabili) nel quadro dell'Accordo Nazionale per l'Incentivazione di uno Stile di Vita Sano (ANISVS): ad esempio, il cliente che consuma troppo caffè può essere disincentivato aumentandone il prezzo, mentre il cliente che consuma troppa poca cioccolata può essere incentivato diminuendolo.

A tal fine, la macchina registra tutti gli acquisti fatti da ogni cliente, con indicazione di data e ora.

Ogni ***strategia di sconto*** specifica:

a) a quale bevanda può essere applicata;
b) su quale intervallo temporale (in ore) valutare i consumi del cliente;
c) quali bevande entrino a far parte della valutazione, con il peso del corrispondente incentivo/disincentivo (un intero positivo o negativo).

La macchina applica la strategia considerando i consumi del cliente nelle ultime T ore (finestra mobile) e calcolando un punteggio P in cui ogni acquisto vale quanto il corrispondente incentivo/disincentivo, o 0 se la bevanda non è considerata dalla strategia. Al termine, il punteggio P diviso per il numero di acquisti N effettuati dal cliente nel periodo determina la percentuale di sconto (se positiva) o incremento di prezzo (se negativa) da applicare al prezzo base.

Ad esempio, la strategia di sconto applicata alla bevanda *Caffè, che* mira a incentivare la cioccolata rispetto al caffè, è così definita:

a) finestra temporale: T = 480 ore (ossia, 20 giorni)
b) incentivi/disincentivi: caffè -2, cioccolata 1

Essa deve intendersi nel senso che, partendo dai consumi del cliente negli ultimi 20 giorni (supponiamo 45 acquisti), ogni caffè toglie 2 punti, ogni cioccolata ne aggiunge 1, mentre tutte le altre bevande valgono 0. Se di quei 45 acquisti 20 sono caffè e solo 5 sono cioccolate, il punteggio finale risulta P = -2 x 20 + 1 x 5 = -35, corrispondente a una percentuale di "incentivo" di -35/45 = -77.78%: essendo negativa, esprime in realtà un *disincentivo*, che si tradurrà in un *aumento del prezzo* del *Caffè* del 77.78%, per questo cliente, rispetto al prezzo base del *Caffè* stesso.

A ogni acquisto la finestra mobile si sposta in avanti, determinando l'esclusione degli acquisti più vecchi dal calcolo. Ad esempio, se il calcolo precedente è stato fatto il 18 giugno, i 45 acquisti sopra citati devono essere stati fatti fra il 30 maggio e il 18 giugno compresi. Se poi il cliente si rifà vivo il 23 giugno, il calcolo dovrà essere rifatto considerando i soli acquisti (che potranno essere in numero qualsiasi) fatti negli ultimi 20gg, ossia fra il 4 e il 23 giugno compresi.

Il file di testo Clienti.txt contiene i clienti registrati, con il relativo ID, mentre il file di testo Bevande.txt specifica per ogni bevanda i prezzi MINIMO, BASE e MASSIMO. Il terzo file di testo, Strategie.txt, specifica le strategie, nel modo più oltre dettagliato. Per mantenere la registrazione di tutti gli acquisti effettuati da tutti i clienti, con data e ora, la macchina gestisce un file binario opportuno, Acquisti.dat (dettagli nella sezione persistenza).

Parte 1 (punti: 20)

Dati (package dentocoffee.model) *(punti: 10)*

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

SEMANTICA:

a) la classe ***Cliente*** (fornita nello start kit) rappresenta un cliente con il suo ID univoco;

b) la classe ***Acquisto*** (fornita) rappresenta uno specifico acquisto di una ben precisa ***Bevanda*** da parte di un ben preciso ***Cliente***, avvenuto in a ben precisa ***java.util.Date***, a un ben preciso prezzo;

c) l'interfaccia ***DentoStrategia*** (fornita) dichiara i metodi ***getNome*** e ***calcolaSconto*** che restituiscono rispettivamente il nome della strategia e lo sconto (un float) da applicare a un dato ***Cliente*** sulla base dei suoi precedenti acquisti (raccolti nell'argomento di tipo ***Collection<Acquisto>***);

[NB: trattandosi di sconto, un valore positivo denota uno sconto, che come tale comporta una riduzione di prezzo, mentre un valore negativo denota un disincentivo, che come tale comporta un aumento di prezzo];

d) la classe ***DefaultDentoStrategia*** (fornita) implementa ***DentoStrategia*** nel caso particolare in cui non vi sia una strategia specifica definita: il nome ritornato è la stringa vuota e lo sconto sempre zero;

e) la classe serializzabile ***DentoStrategiaConfigurabile*** (da realizzare) deriva da ***DentoStrategiaBase*** (nello start kit) che, a sua volta, implementa ***DentoStrategia*** secondo le specifiche dettagliate nel dominio del problema. Il costruttore riceve tre argomenti: il nome della strategia, una mappa (***String***, ***Integer***) che associa al **nome** di una bevanda il punteggio corrispondente (non tutte le bevande sono necessariamente presenti!) e la larghezza della finestra temporale in ore;

IMPORTANTE: **affinché i test operino correttamente, è *INDISPENSABILE*** ottenere l'oggetto ***Date*** contenente la data/ora corrente chiamando il metodo ***getNow*** contenuto in ***DentoStrategiaBase.***

f) la classe serializzabile ***Bevanda*** (da realizzare) rappresenta una specifica bevanda: prevede un unico costruttore che riceve quattro argomenti (nome, prezzo min, prezzo base, prezzo max), i metodi accessor ***getNome***, ***getPrezzoBase***, ***getPrezzoMin***, ***getPrezzoMax***, la coppia di accessor ***setStrategia***/***getStrategia*** e il metodo ***getPrezzo***. In particolare:

- ***getStrategia***: restituisce sempre una strategia valida (un riferimento non nullo), al limite la strategia di default, pertanto se la strategia attualmente impostata è nulla allora restituisce un'istanza di ***DefaultDentoStrategia***;
- ***getPrezzo***: dato un ***Cliente*** e la collezione dei suoi precedenti acquisti, restituisce il prezzo da applicargli per la bevanda in oggetto; per fare ciò recupera la strategia (metodo accessor ***getStrategia***) tramite la quale ottiene lo sconto con il quale calcola il prezzo, se il prezzo è fuori dal range ammissibile allora il prezzo sarà pari al margine superato.

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

Persistenza (package dentocoffee.persistence) *(punti 10)*

Come già anticipato, il file di testo Clienti.txt contiene i clienti registrati, uno per riga: nell'ordine sono presenti l' ID univoco, la denominazione del cliente (anche più parole): i vari campi sono separati fra loro da tabulazioni.

```
10          Guido La Vespa
20          Remo La Barca
…
```

Analogamente, il file di testo Bevande.txt contiene le varie bevande, una per riga: per ciascuna sono definiti il nome e i tre prezzi min/base/max, separati da una o più tabulazioni:

```
Cioccolata          0.10  0.30  1
Caffè               0.20  0.30  2
Latte               0.20  0.20  0.20
…
```

Infine, il file Strategie.txt specifica le strategie: ognuna è descritta da un blocco su più righe nel seguente formato:

```
STRATEGIA   Cioccolata   SU   480 ORE
caffè = -1, cioccolata = 2,  tè al limone = 1
FINE STRATEGIA
…
```

Ogni blocco è così strutturato:

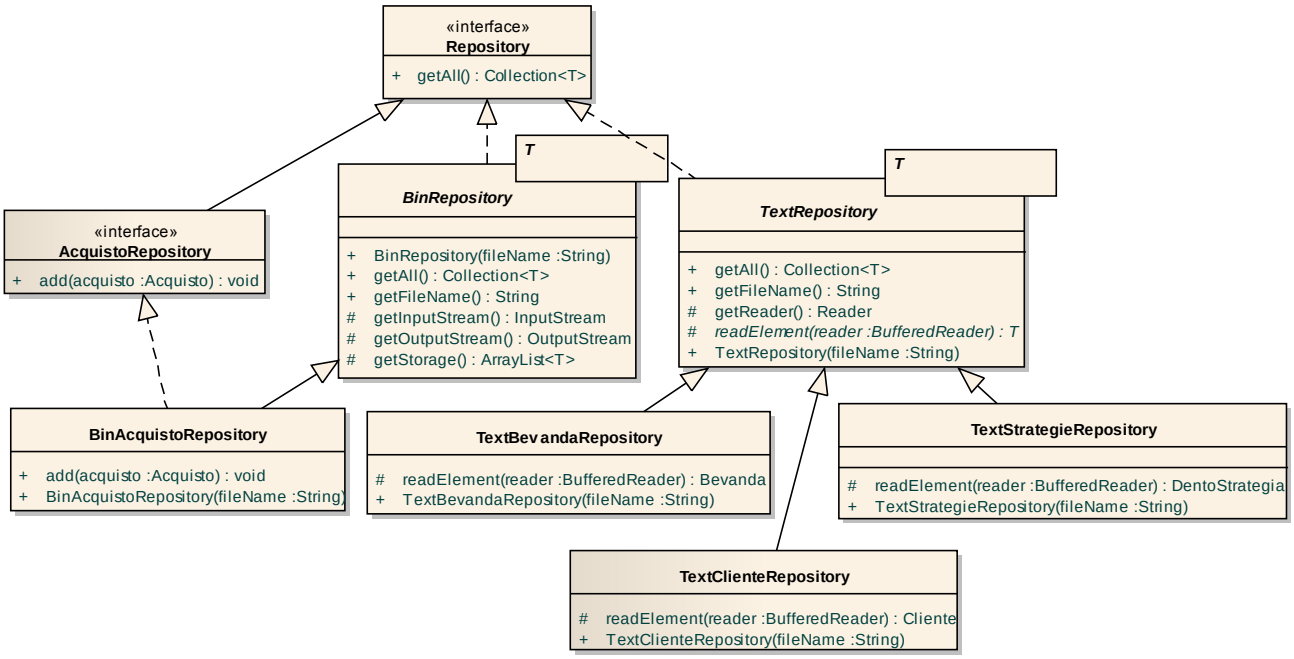
- la prima riga contiene, separati da tabulazioni, la parola chiave "STRATEGIA", un nome che può contenere spazi, la parola chiave "SU", un numero intero positivo, la parola chiave "ORE";
- la seconda riga contiene l'elenco delle bevande incentivate/disincentivate, costituito da una serie di voci separate da virgole: ogni voce comprende il nome della bevanda (che può contenere spazi), il simbolo "=", il peso intero dell'incentivo. [ricorda che le bevande non citate si suppongono avere peso zero];
- la terza riga contiene sempre la frase "FINE STRATEGIA"

Infine, come accennato sopra, il file binario Acquisti.dat registra tutti gli ***Acquisto*** effettuati da tutti i clienti.

L'architettura software si basa sul concetto di ***REPOSITORY,*** inteso come gestore di un certo tipo di dati che nasconde il tipo di file (binario o di testo) utilizzato per l'archiviazione, incapsula l'apertura e la chiusura dei file e offre ai suoi clienti metodi di alto livello con cui operare sui dati indipendentemente dalla realizzazione fisica.

Tale astrazione è catturata dall'interfaccia generica ***Repository<T>,*** il cui unico metodo ***getAll*** restituisce gli elementi letti sotto forma di ***Collection<T>***.
Essa è poi *parzialmente concretizzata* dalle due classi astratte ***BinRepository*** e ***TextRepository*** rispettivamente nel caso di *repository su file binario* e *repository su file di testo*.
Particolarità: poiché il repository degli acquisti, a differenza degli altri, deve supportare anche la scrittura di dati, è introdotta una specifica interfaccia ***AcquistoRepository*** che aggiunge la dichiarazione del metodo ***add***.

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



SEMANTICA:

a) come già anticipato, l'interfaccia generica ***Repository<T>*** dichiara il metodo ***getAll*** che restituisce gli elementi letti sotto forma di ***Collection<T>***; la sotto-interfaccia ***AcquistoRepository*** dichiara anche il metodo ***add(Acquisto)***;

b) la classe astratta ***TextRepository<T>*** (fornita) realizza parzialmente il caso in cui il repository sia un file di testo, memorizzando internamente i dati in una *ArrayList<T>*. Il costruttore effettua la lettura di tutti gli elementi contenuti nel file, che saranno poi forniti al cliente da ***getAll.*** L'unico metodo astratto è il metodo protetto ***readElement*** che restituisce un singolo elemento di tipo T estratto dal ***BufferedReader*** ricevuto come argomento;

c) dualmente, la classe astratta ***BinRepository<T>*** (fornita) realizza il caso in cui il repository sia un file binario: anche qui il costruttore effettua in blocco la lettura di tutti gli elementi contenuti nel file, che saranno poi forniti al cliente da ***getAll.*** Non vi sono metodi astratti: la classe è astratta solo per impedirne l'istanziazione diretta.

d) la classe ***BinAcquistoRepository*** **(da realizzare)** specializza ***BinRepository<Acquisto>*** implementando al contempo l'interfaccia ***AcquistoRepository***; poiché ***getAll*** è ereditato dalla classe base, questa classe deve implementare in realtà soltanto il metodo ***add(Acquisto)***, che aggiunge un ***Acquisto*** al repository: tale operazione prevede sia l'aggiunta dell'acquisto nella collezione degli acquisti (recuperabile tramite il metodo ***getStorage*** ereditato dalla classe base), sia la riscrittura globale del file binario in modo compatibile con la successiva ri-lettura effettuata dal ***getAll*** ereditato (si veda il codice della classe base) [SUGGERIMENTO: prevedere un metodo privato ***writeAll***];

e) le classi ***TextBevandaRepository*** e ***TextClienteRepository*** (fornite) specializzano ***TextRepository<T>*** nei due casi delle bevande e dei clienti, definendo il metodo ***readElement*** nel modo adatto ai rispettivi file;

f) la classe ***TextStrategieRepository*** **(da realizzare)** specializza ***TextRepository<T>*** nel caso delle strategie: l'implementazione del metodo ***readElement*** dovrà dunque essere adatta al formato del file Strategie.txt.

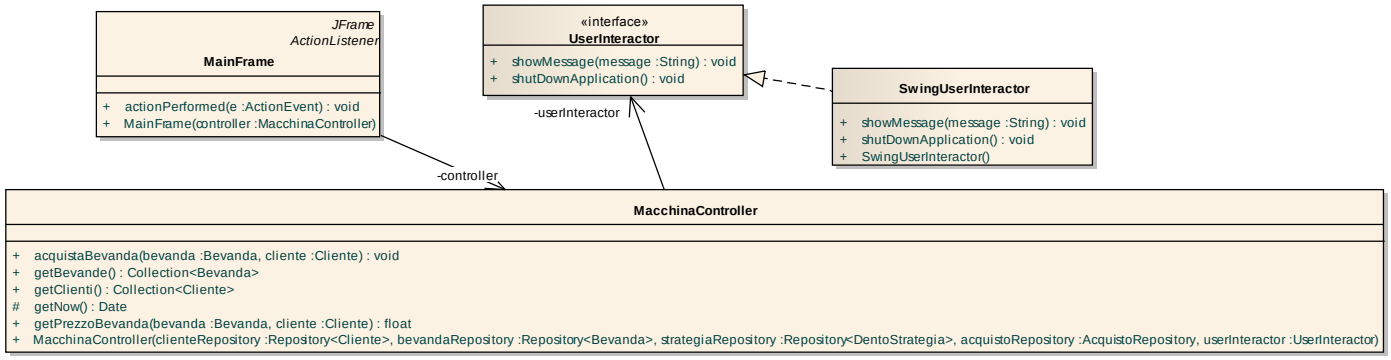
Lo Start Kit contiene anche i test (da includere nel progetto) per verificare il funzionamento di questa classe.

Parte 2 (punti: 10)

L'interfaccia grafica deve simulare una macchina erogatrice di bevande, in modo simile (non necessariamente identico) all'esempio mostrato di seguito.

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

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



Il controller della macchina è fornito già implementato: è quindi richiesto di realizzare unicamente il MainFrame.

SEMANTICA:

a) L'interfaccia ***UserInteractor*** (fornita), implementata dalla classe ***SwingUserInteractor*** (fornita), consente di mostrare un messaggio all'utente (metodo ***showMessage***) o di terminare l'applicazione (***shutDownApplication***);

b) La classe ***MacchinaController*** (fornita) presenta un costruttore con quattro argomenti (i tre repository e uno ***UserInteractor***) e definisce i seguenti metodi:

- ***getBevande*** e ***getClienti*** restituiscono rispettivamente una ***Collection<Bevanda>*** e una ***Collection<Cliente>***;
- ***getPrezzoBevanda*** restituisce il prezzo da applicare per una data ***Bevanda*** a un dato ***Cliente***, tenendo conto della strategia di sconto applicata, dello storico degli acquisti e dei prezzi minimi e massimi per tale bevanda;
- ***acquistaBevanda*** effettua l'acquisto della ***Bevanda*** specificata da parte del ***Cliente*** specificato, aggiornando il repository degli acquisti.

La classe ***MainFrame*** **(da realizzare)** costituisce la GUI dell'applicazione: vi sono due combo indipendenti con, rispettivamente, l'elenco dei clienti e delle bevande, una label per l'indicazione del prezzo corrente della bevanda e un pulsante **Acquista** (Fig. 1).

Ogni volta che si preme il pulsante si simula un acquisto: il prezzo mostrato varia di conseguenza, secondo l'algoritmo (Fig. 2).

Naturalmente, se si cambia bevanda (Fig. 3) il prezzo mostrato si adegua; lo stesso accade effettuando ulteriori acquisti (Fig. 4) o se si cambia cliente (Fig. 5, Fig. 6).

ATTENZIONE: i prezzi mostrati nelle figure seguenti sono puramente indicativi, in quanto dipendono dalla specifica sequenza di acquisti seguita.

Figura 1

Figura 2

Figura 3

Figura 4

Figura 5

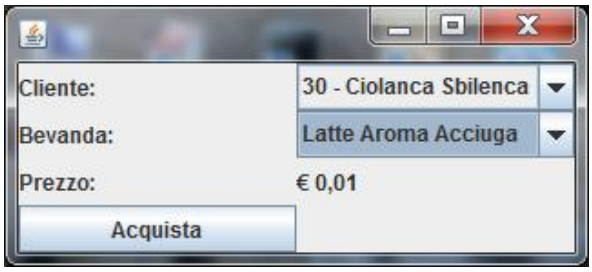


Figura 6