ESAME DI FONDAMENTI DI INFORMATICA T-2 del 07/09/2016
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 e CARTELLA: CognomeNome-matricola (es. RossiMario-0000123456)
NOME ZIP DA CONSEGNARE: CognomeNome-matricola.zip (es. RossiMario-0000123456.zip)

Il corriere espresso *ZShipments* ha richiesto lo sviluppo di un'applicazione da dare al proprio personale per tracciare in tempo reale le consegne (o mancate consegne) dei propri pacchi.

DESCRIZIONE DEL DOMINIO DEL PROBLEMA.
L'applicazione mostra la lista dei pacchi da consegnare (*Shipment*). Ogni volta che si tenta una consegna (*Delivery*), essa può avere successo (se si trova il destinatario in casa: *SucceededDelivery*) oppure fallire (*FailedDelivery*): in questo caso dovrà essere ritentata successivamente una nuova consegna. Di conseguenza, ogni *Shipment* può essere associato a zero o più *FailedDelivery* ma a una sola *SucceededDelivery.*

Il file di testo Shipments.txt contiene l'elenco delle spedizioni, nel formato più oltre specificato.

Parte 1 (punti 17)

Dati (namespace zs.model) *(punti: 6)*

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

SEMANTICA:

a) la classe ***Recipient*** (fornita) rappresenta il destinatario della spedizione, con le sue proprietà;

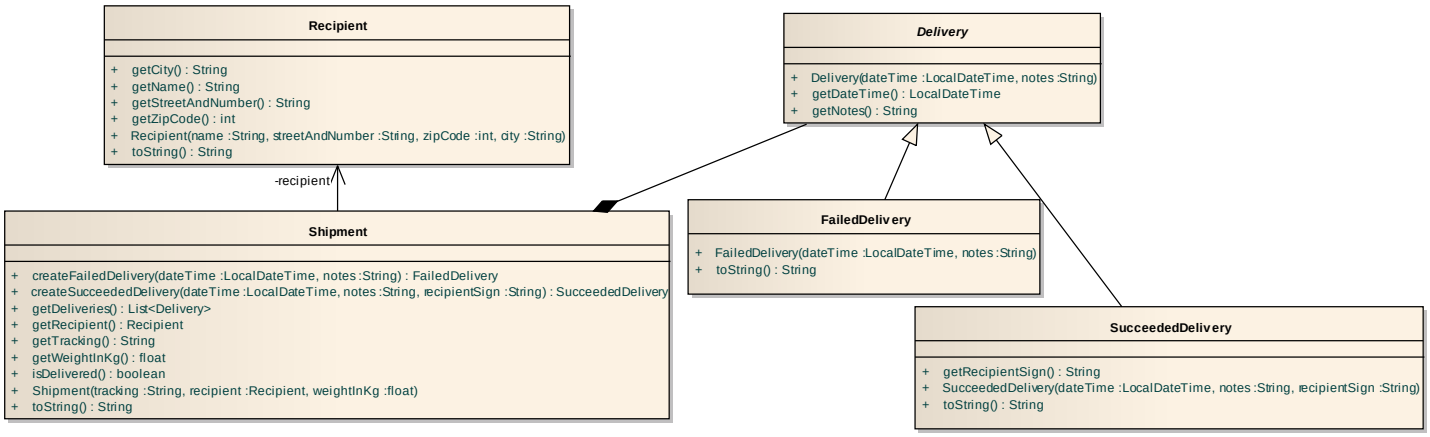
b) la classe astratta ***Delivery*** (fornita) fattorizza le proprietà comuni alle consegne riuscite e fallite;

c) la classe ***SucceededDelivery*** (fornita) rappresenta la consegna riuscita;

d) la classe ***FailedDelivery*** (fornita) rappresenta la consegna fallita;

e) **la classe *Shipment* (da realizzare)** rappresenta la spedizione, caratterizzata da *stringa di tracking, destinatario* (un ***Recipient***) *e peso* (un valore *float*), recuperabili tramite appositi *accessor*. Il metodo ***getDeliveries*** restituisce la lista (eventualmente vuota) di ***Delivery*** contenente tutte le consegne effettuate (finora) per questa spedizione, mentre il metodo ***isDelivered*** è vero solo se la spedizione è stata consegnata con esito positivo, ossia se nell'elenco delle consegne ne esiste una – l'ultima – che sia una ***SucceededDelivery***. I due metodi ***createSucceededDelivery*** e ***createFailedDelivery*** aggiungono alla lista delle consegne effettuate rispettivamente una nuova consegna riuscita / fallita, con i rispettivi dati (orario, note, e, nel solo caso della consegna riuscita, la firma di chi ha ricevuto il pacco). Il metodo ***toString*** si rimappa su ***toString*** di ***Recipient***. Ogni metodo lancia opportune eccezioni (***IllegalArgumentException*** o ***InvalidOperationException***), con adeguato e distinto messaggio d'errore, rispettivamente nel caso in cui i parametri non siano corretti o lo stato dell'istanza in ***createSucceededDelivery*** o ***createFailedDelivery*** non consenta il completamento dell'operazione; in particolare:

- i parametri di tipo riferimento del costruttore devono essere non nulli (e, nel caso della stringa di tracking, anche non vuota); il peso della spedizione, che è un float, dev'essere maggiore di zero;
- fra i parametri di ***createSucceededDelivery***, la data/ora deve essere non nulla e la firma deve essere non nulla e non vuota, mentre non si impongono vincoli sulla nota;
- fra i parametri di ***createFailedDelivery***, la data/ora deve essere non nulla, mentre non si impongono vincoli sulla nota.



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

Persistenza (namespace zs.persistence) *(punti 11)*

Come già anticipato, il file di testo Shipments.txt contiene le consegne da effettuare, una per riga. Ogni riga contiene una serie di dati: nella parte iniziale di ogni riga, spazi o tabulazioni fungono da separatori; dal carattere '@' in poi e fino a fine riga, però, l'unico separatore ammesso diviene il solo carattere '@', in quanto questi dati possono contenere spazi o tabulazioni al loro interno. Più precisamente, una riga è composta da *tracking, peso e destinatario*, separati da spazi o tabulazioni; il destinatario però non è un campo atomico, ma è ulteriormente composto da quattro elementi – *cognome e nome, indirizzo, cap e città* – separati appunto dal carattere '@'.

FILE Shipments

```
132UX321IERT88 1.10 Gabriele Zannoni@Via del Pero, 21@48100@Ravenna
...
```

Un altro file di testo, Deliveries.txt (inizialmente non esistente), tiene traccia delle consegne (riuscite e fallite). La gestione di tale file è completamente delegata a un opportuno ***ShipmentRepository***, di cui si fornisce anche l'implementazione ***MyShipmentRepository:*** esso gestisce automaticamente consegne e spedizioni appoggiandosi ad uno ***ShipmentsReader*** e un ***DeliveriesWriter***, ricevuti nel costruttore (dettagli nel seguito). ***ShipmentRepository*** espone i seguenti metodi:

- ***getAll***, che restituisce la lista di tutti gli ***Shipment***;
- ***getDelivered***, che restituisce la lista degli ***Shipment*** che sono stati consegnati con successo;
- ***getNonDelivered***, che restituisce la lista degli ***Shipment*** che non sono stati consegnati;
- ***getByTracking***, che restituisce, sotto forma di ***Optional<Shipment>***, lo ***Shipment*** corrispondente al *tracking* passato come argomento (se esiste);
- ***update***, che aggiorna il file delle consegne (purché lo ***Shipment*** passato come argomento sia presente nel file).

Le due interfacce ***ShipmentsReader*** e ***DeliveriesWriter*** dichiarano rispettivamente:

- la prima, il metodo ***read***, che legge da un ***BufferedReader*** una lista di ***Shipment***
- la seconda, il metodo ***write***, che scrive su un ***BufferedWriter*** tutte le Delivery contenute nella *collection* di ***Shipment***.

La classe *MyShipmentsReader* (da realizzare) implementa la prima interfaccia, lanciando ***IOException*** in caso di errore di lettura fisico e ***BadFileFormatException*** (fornita), con opportuno messaggio d'errore specifico, nel caso di errore nel parsing della riga.

La classe *MyDeliveriesWriter* (da realizzare) implementa la seconda interfaccia, lanciando ***IOException*** in caso di errore di scrittura. Questa classe deve generare il file Deliveries.txt, inizialmente non esistente, scrivendo su esso una riga per ogni ***Delivery*** con formato distinto fra consegne fallite o riuscite. Più precisamente:

- per le consegne fallite, dopo la parola "Failed", vengono stampati solo il tracking, la data e l'ora e le note;

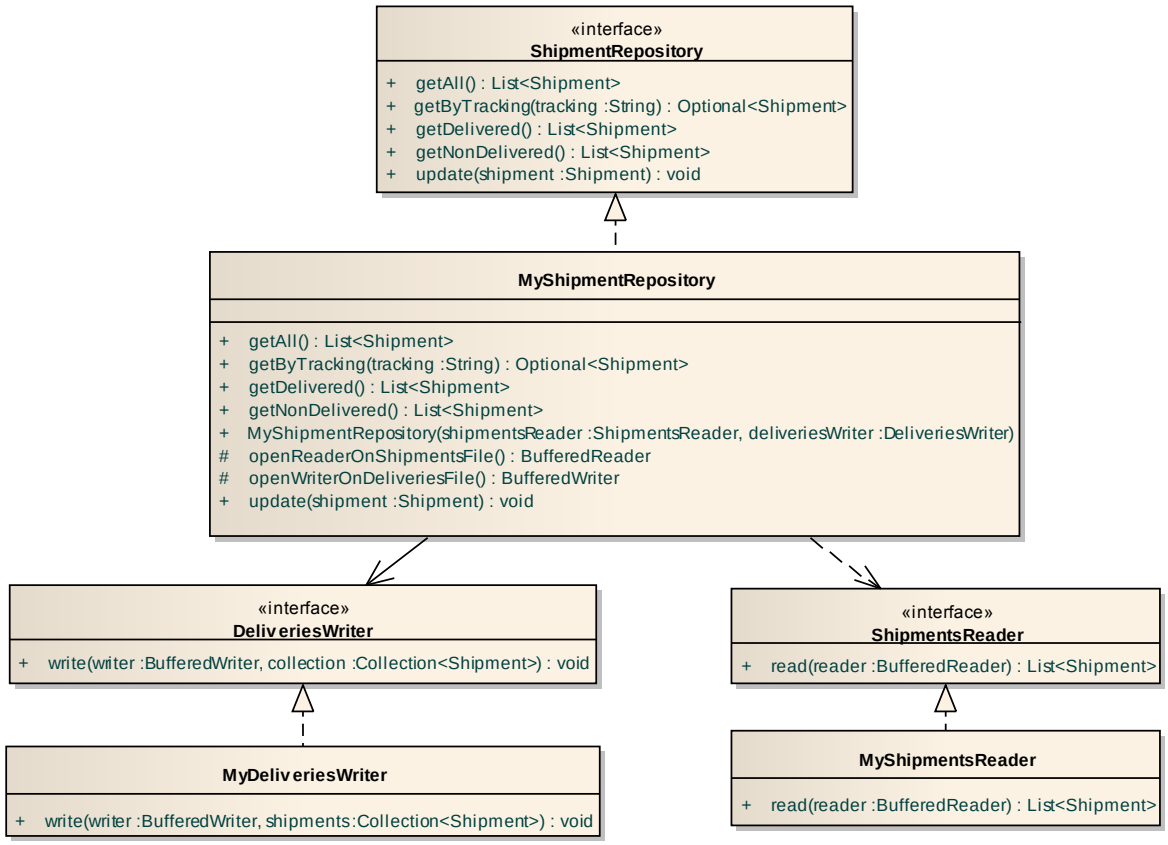
- per le consegne riuscite, dopo la parola “Succeeded”, vengono stampati il tracking, la data e l’ora, *la firma della persona che ha ricevuto il pacco* e le note;

In entrambi i casi, le informazioni relative a data/ora sono scritte mediante l’uso del *formatter* ***DateTimeFormatter.ISO_LOCAL_DATE_TIME***; i dati sono sempre separati solo dal carattere “;”, come nell’esempio che segue:

FILE Deliveries

```
Failed;442HHASD882233;2016-08-06T22:08:36.242;non in casa
Succeeded;442HHASD882233;2016-08-06T22:09:01.103;Nonna Piera;trovato finalmente
...
```

Per semplicità, ad ogni richiesta di scrittura il file viene interamente rigenerato e, per ogni ***Shipment*** contenuta nella collezione ricevuta come argomento, vengono scritte tutte le ***Delivery*** in essa contenute.



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

Parte 2 (punti 13)

IMPORTANTE: per consentire il test della GUI anche nel caso di *reader* non funzionanti, è fornita la classe **GUITestProgram** che replica **Program** utilizzando però dati fissi pre-cablati al posto di quelli letti da file.

Controller (namespace zs.ui) *(punti 5)*

L’interfaccia ***Controller*** (fornita) dichiara quattro metodi che devono essere implementati dalla classe ***MyController*** **(da realizzare):**

- il costruttore riceve lo ***ShipmentRepository*** su cui agire, nonché uno ***UserInteractor*** che consente di interagire con l’utente chiedendo input o mostrando messaggi di avviso;
- ***markFailedDelivery***, tramite il repository, annota sull’apposito file la fallita consegna dello shipment dato; per fare ciò, occorre chiedere all’utente eventuali note da apporre (***UserInteractor.requestUserNotes***), indi creare una consegna fallita mediante l’apposito metodo di ***Shipment*** (passando la data/ora corrente e la nota precedentemente ottenuta) e, infine, aggiornare il file (metodo ***Repository.update***); in caso di problemi nella

scrittura del file, occorre informare l'utente notificandogli il messaggio contenuto nell'eccezione lanciata dal metodo di aggiornamento (***UserInteractor.somethingWentWrongWhileSaving***);

- ***markSucceededDelivery***, tramite il *repository*, annota sull'apposito file la riuscita consegna dello ***Shipment*** dato; per fare ciò, occorre chiedere all'utente prima eventuali note da apporre, poi la firma del destinatario (***UserInteractor.requestRecipientSign***) e con queste creare una consegna avvenuta con successo mediante l'apposito metodo di ***Shipment***; per il resto, procedere come sopra;
- ***getDelivered***, che restituisce la lista degli ***Shipment*** consegnati con successo (ottenuti dal *repository*);
- ***getNonDelivered***, che restituisce la lista degli ***Shipment*** che non sono stati consegnati (ottenuti dal *repository*).

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

Interfaccia utente (namespace zs.ui) *(punti 8)*

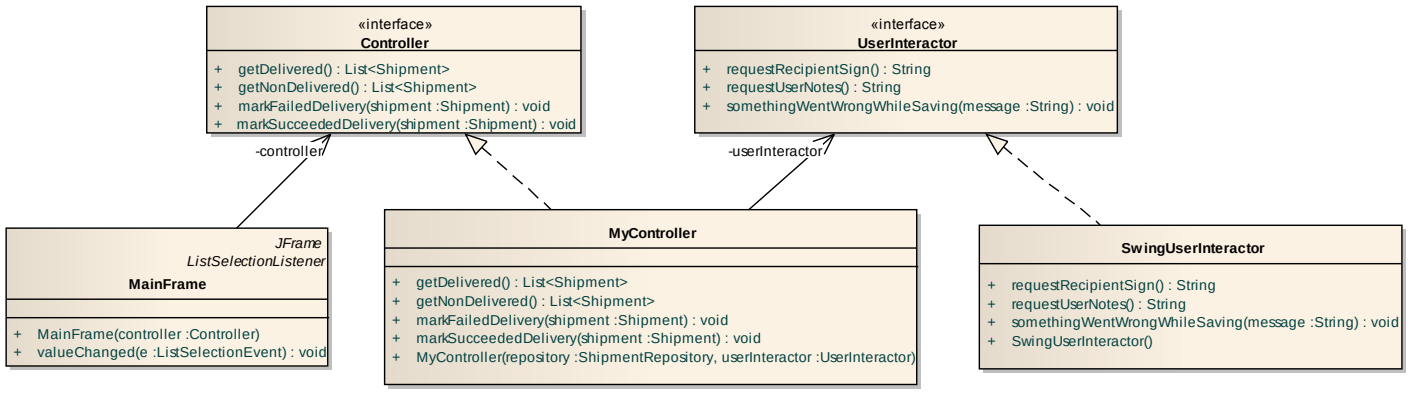
L'interfaccia utente deve essere simile (non necessariamente identica) all'esempio mostrato in figura.

La classe *MainFrame* (da realizzare) realizza la finestra principale, che è organizzata con ***BorderLayout***. Nell'area centrale è posto un ***JPanel*** organizzato con ***GridLayout*** con una riga e due colonne (due celle in tutto); nelle due celle ci sono due ***EasyJList*** (un componente derivato da ***JList*** che ne semplifica l'utilizzo, definendo tre semplici metodi ***addElement***, ***removeElement*** e ***clearElements*** – vedere la documentazione Javadoc inserita nel codice per i dettagli) – una per gli ***Shipment***, a sinistra, e una per le consegne (***Delivery***) relative allo ***Shipment*** selezionato, a destra. Nell'area inferiore di ***MainFrame*** è posto un altro pannello, organizzato con ***FlowLayout,*** che contiene due ***JButton***, etichettati rispettivamente **Successful Delivery** e **Failed Delivery**, che consentono al corriere di inserire i dati di una nuova consegna *riuscita* o *fallita*, rispettivamente. [Estetica: è gradito, ma non richiesto, che i due pulsanti abbiano egual dimensione, come nelle figure sotto riportate].

Inizialmente (Fig. 1) la lista di sinistra mostra tutte le spedizioni poiché nessuna è ancora stata consegnata. Selezionandone una (Fig. 2) e premendo sul pulsante **Successful Delivery,** compaiono in sequenza due *dialog* generate mediante ***UserInteractor*** (Figg. 2 e 3) che consentono di inserire le note (***UserInteractor.requestUserNotes***, invocato dal controller) e la firma della persona che riceve il pacco (***UserInteractor.requestRecipentSign***, invocato dal controller); come effetto di ciò, il pacco appena consegnato scompare dalla lista degli ***Shipment*** da spedire (perché è appena stato consegnato!) (Fig. 5).

Naturalmente, se viene invece premuto il pulsante **Failed Delivery** (non mostrato), compare soltanto la prima *dialog*, dato che in una consegna fallita la persona che ha ricevuto il pacco non esiste; in tal caso, il pacco logicamente NON scompare dalla lista degli ***Shipment***, dato che dovrà essere ri-consegnato successivamente.

Alla selezione di una spedizione, nella ***EasyJList*** di destra compaiono tutte le ***Delivery*** corrispondenti: esse non potranno che essere tutte fallite, poiché se il pacco fosse stato consegnato non sarebbe stato selezionabile.



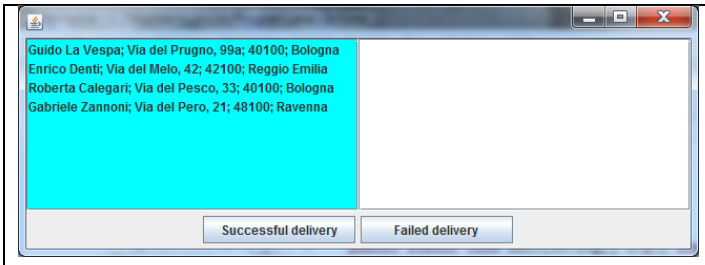


Fig. 1

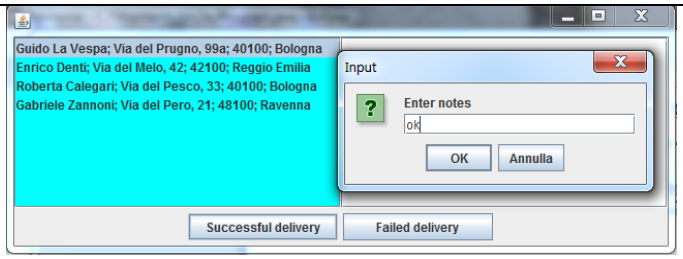


Fig. 2

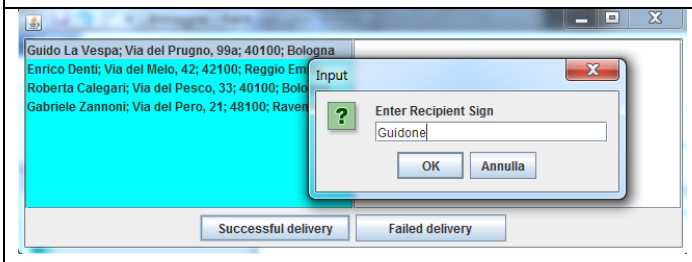


Fig. 3

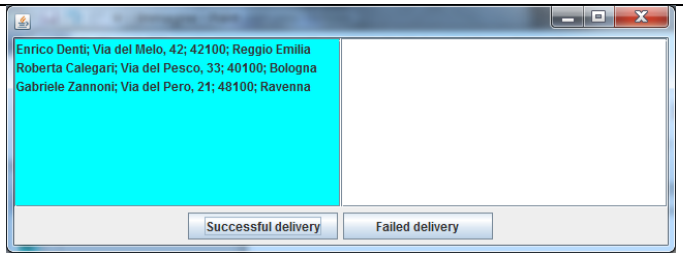


Fig. 4