ESAME DI FONDAMENTI DI INFORMATICA T-2 del 27/06/2016
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)

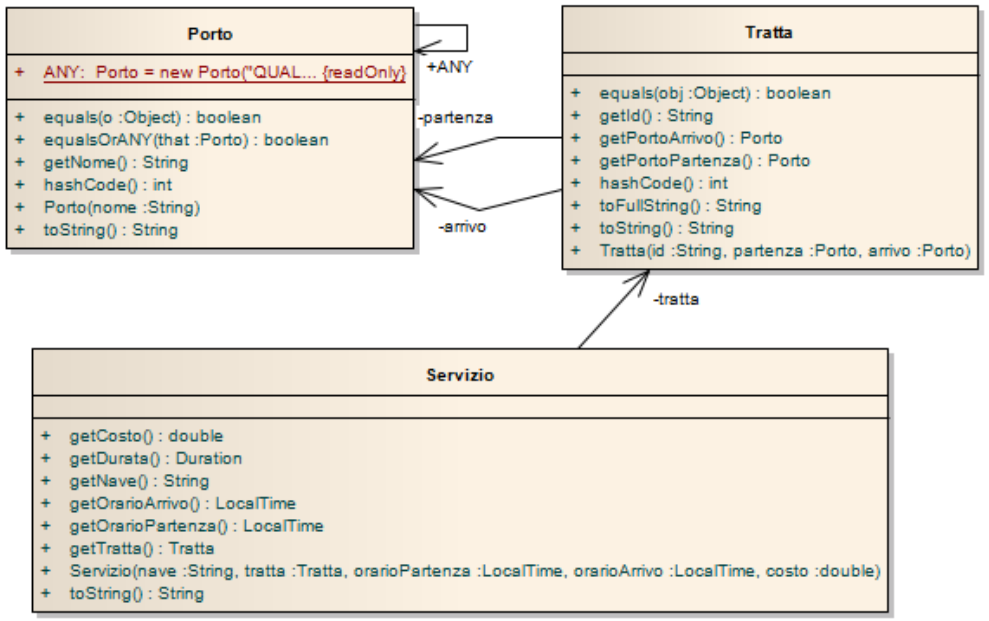
La compagnia di navigazione *TeethFerries* ha richiesto un'applicazione che illustri le soluzioni di viaggio fra i due porti scelti dall'utente (che possono essere anche "qualsiasi"), come più oltre mostrato.

> DESCRIZIONE DEL DOMINIO DEL PROBLEMA.
> Un traghetto effettua servizio su una singola *tratta,* senza fermate intermedie: una tratta è definita dai due porti di partenza e arrivo. Su quella tratta possono essere disponibili più *servizi,* caratterizzati ciascuno dal nome della nave che lo effettua, dagli orari di partenza e arrivo e naturalmente dal relativo costo.

Parte 1 (punti: 16)

Dati (namespace teethferries.model) *(punti: 7)*

I dati relativi ai traghetti e alle tratte sono organizzati in accordo al diagramma UML in figura.



SEMANTICA:

a) La classe ***Porto*** (fornita nello start kit) rappresenta un porto col proprio nome univoco, recuperabile con l'apposito *accessor*; sono opportunamente ridefiniti ***toString***, ***equals*** e ***hashcode.*** La costante pubblica ***ANY*** rappresenta il porto "qualsiasi" (utile nelle *combobox* della GUI), mentre il metodo ausiliario ***equalsOrANY*** è vero se l'argomento è uguale al Porto corrente, oppure uno dei due è ***ANY***.

b) La classe ***Tratta*** (fornita nello start kit) rappresenta una tratta, caratterizzata da un identificatore univoco e dai due porti di partenza e arrivo, recuperabili tramite appositi *accessor*: ovviamente, anche qui ***toString*** è opportunamente ridefinito; in più, il metodo ***toFullString*** produce una stringa più completa e dettagliata.

c) La classe ***Servizio* (da realizzare)** rappresenta un servizio svolto su una data tratta, a specifici orari, da una ben precisa nave. Il costruttore riceve il nome della nave, una ***Tratta***, gli orari di partenza e arrivo e il costo del servizio; se uno degli argomenti è *null* o il costo è negativo (o 0) dev'essere lanciata una ***IllegalArgumentException*** con opportuno messaggio. Devono inoltre essere disponibili i seguenti metodi:

   - gli accessor ***getNave***, ***getTratta***, ***getOrarioPartenza***, ***getOrarioArrivo***;
   - ***getDurata*** per ottenere la durata del servizio stesso;

- ***toString*** ridefinito appoggiandosi all'omonimo metodo di ***Tratta*** in modo da produrre una stringa della forma "*nave* da *porto* a *porto hh:mm-hh:mm*", dove *hh:mm* è un orario correttamente formattato con un *formatter* ottenuto mediante il metodo *factory* ***DateTimeFormatter.ofPattern*** specificando il pattern ***"HH:MM"***.

Persistenza (namespace teethferries.persistence) *(punti: 9)*

Il file di testo Servizi.txt contiene la descrizione dei servizi offerti, uno per riga: ogni riga contiene la tratta servita, gli orari di partenza e arrivo (usare il *formatter* ISO_LOCAL_TIME), la nave che la effettua e il relativo costo in euro, separati da punti e virgola.

A loro volta, le tratte sono descritte nel file di testo Tratte.txt, una per riga: ogni riga contiene semplicemente un identificativo univoco, il porto di partenza e il porto di arrivo, separati da tabulazioni.

ESEMPIO DI FILE servizi.txt

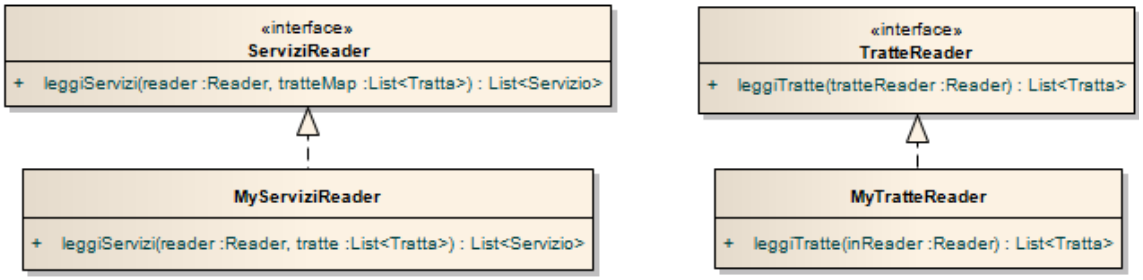
```
Sardegna1;7:30;12:30;Dente Appuntito;€ 25,00
...
```

ESEMPIO DI FILE tratte.txt

```
Sardegna1   Livorno          Olbia
Sardegna5   Civitavecchia    Cagliari
Sardegna7   Genova           Olbia
…
Corsica2    Savona           Calvi
Corsica3    Savona           Bastia
…
```

a) L'interfaccia **TratteReader** (fornita nello start kit) dichiara il metodo ***leggiTratte*** che, dato un **Reader**, legge e restituisce una lista di tratte. La classe ***MyTratteReader*** (**da realizzare**) deve implementare **TratteReader** nel caso specifico del formato sopra specificato: in caso di errore nel formato del file, il metodo di lettura deve incapsulare ogni problema in un'opportuna ***MalformedFileException***.

b) L'interfaccia **ServiziReader** (fornita) dichiara il metodo ***leggiServizi*** che, dato un **Reader** e una lista di tratte ottenuta come sopra, legge e restituisce una lista di servizi. La classe ***MyServiziReader*** (**da realizzare**) implementa **ServiziReader** nel caso specifico del formato sopra specificato: in caso di errore nel formato del file, anche questo metodo di lettura deve incapsulare ogni problema in un'opportuna ***MalformedFileException***.



Parte 2 (punti: 14)

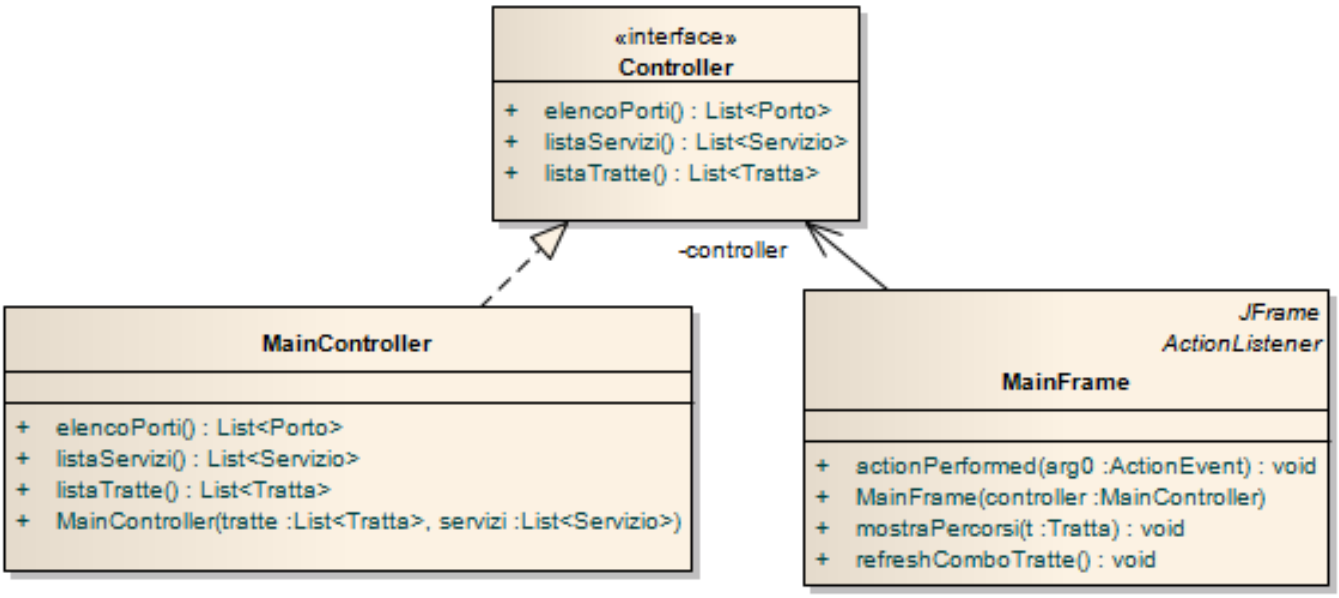
IMPORTANTE: per consentire il test della GUI anche nel caso di reader non funzionanti, è fornita la classe **GUITestProgram** che replica il funzionamento della classe **Program** utilizzando dati fissi pre-cablati al posto di quelli letti da file.

Controller (namespace teethferries.controller) *(punti: 6)*

L'interfaccia **Controller** (fornita) dichiara tre metodi: ***elencoPorti***, che restituisce la lista di tutti i porti con all'inizio il porto "qualsiasi", ***listaTratte*** e ***listaServizi***, che restituiscono la lista di tutte le tratte e di tutti i servizi, rispettivamente.

La classe ***MainController*** **(da realizzare)** implementa **Controller**: il costruttore riceve in ingresso la lista di tratte e la lista di servizi e lancia le necessarie eccezioni nel caso in cui i parametri siano *null*.

Nota: il metodo ***elencoPorti*** deve restituire l'elenco di tutti i porti contenuti nelle tratte senza riportare duplicati con, in più, il porto qualsiasi (***Porto.ANY***). [SUGGERIMENTO: per fare ciò, tenendo conto del fatto che ***Porto*** ridefinisce i metodi ***hashCode*** e ***equals***, è possibile usare in modo opportuno un ***HashSet***…]



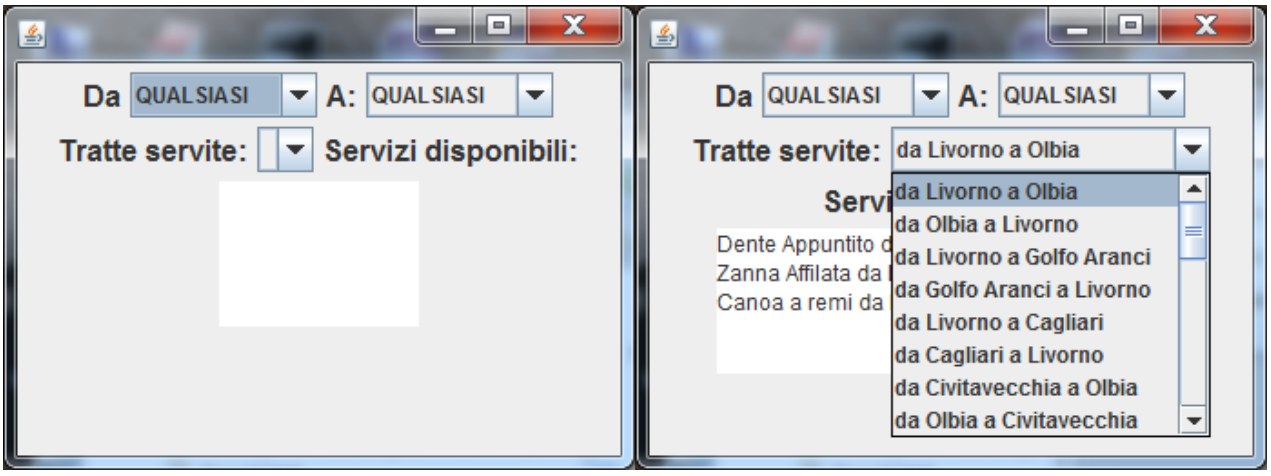
GUI (namespace teethferries.ui) *(punti: 8)*

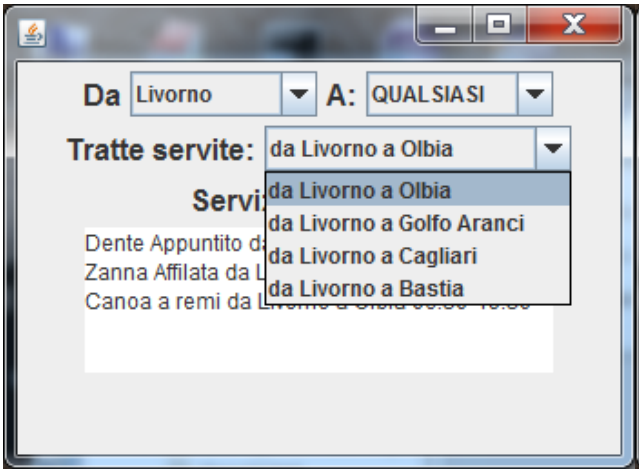
L'interfaccia utente deve essere simile (non necessariamente identica) all'esempio mostrato in figura. La classe ***Program*** (fornita ma **non mostrata** nel diagramma UML) contiene il *main* di partenza dell'intera applicazione.

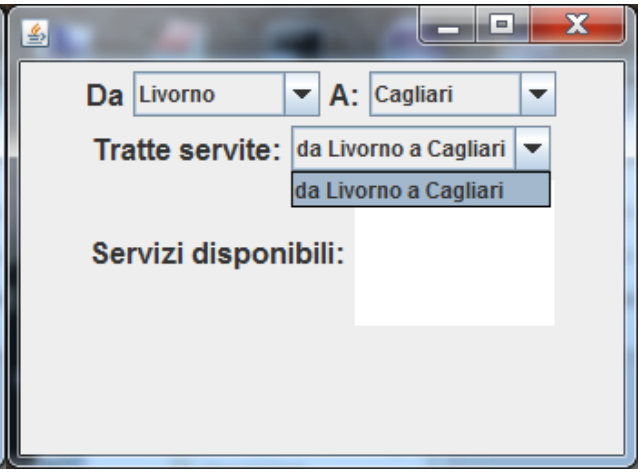
La classe ***MainFrame*** **(da realizzare)** realizza la finestra principale, che deve consentire all'utente innanzitutto di selezionare i porti di partenza e arrivo dalle apposite *combobox* (Fig. 1).

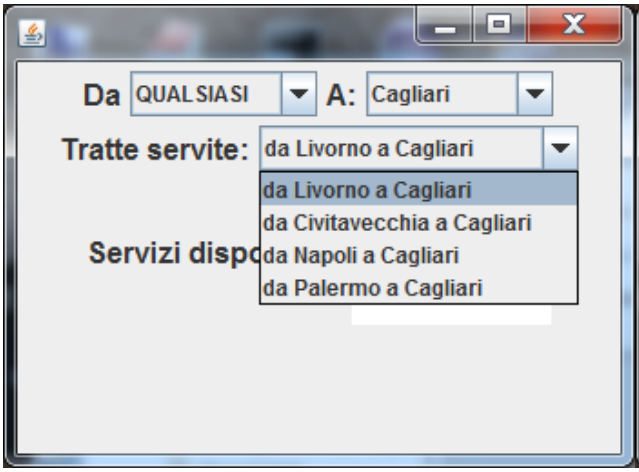
Appena uno di essi viene selezionato, occorre aggiornare la sottostante combo delle tratte (Fig. 2) in modo che contenga solo le tratte corrispondenti ai porti selezionati: ovviamente, se il porto di partenza e/o di arrivo è "qualsiasi", devono essere incluse tutte le tratte con qualsiasi porto di partenza e/o di arrivo (Figg. 3 e 4).

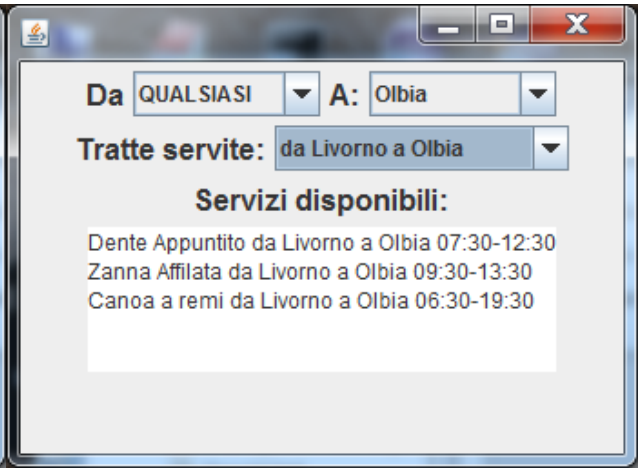
Quando l'utente, poi, seleziona una delle tratte, la *textarea* sottostante deve mostrare tutti i servizi attivi su tali tratte (Figg. 5 e 6)




Da Livorno A: QUALSIASI
Tratte servite: da Livorno a Olbia
da Livorno a Olbia
da Livorno a Golfo Aranci
da Livorno a Cagliari
da Livorno a Bastia
Dente Appuntito d
Zanna Affilata da L
Canoa a remi da


Da Livorno A: Cagliari
Tratte servite: da Livorno a Cagliari
da Livorno a Cagliari
Servizi disponibili:


Da QUALSIASI A: Cagliari
Tratte servite: da Livorno a Cagliari
da Livorno a Cagliari
da Civitavecchia a Cagliari
da Napoli a Cagliari
da Palermo a Cagliari
Servizi dispo


Da QUALSIASI A: Olbia
Tratte servite: da Livorno a Olbia
Servizi disponibili:
Dente Appuntito da Livorno a Olbia 07:30-12:30
Zanna Affilata da Livorno a Olbia 09:30-13:30
Canoa a remi da Livorno a Olbia 06:30-19:30