Documentazione Object Orientation

Compagnie di navigazione

Anno accademico 2023-2024

Professori:
Silvio Barra
Porfiro Tramontana

Alunni:
Francesco Sorrentino N86004817
Antonio Paudice N86004566
Deyvid Dimitrov Manolov N86004796

Indice

1 Dominio del problema **2**
1.1 Analisi dei requisiti 2
1.2 Diagramma del Dominio del Problema 4

2 Dominio della Soluzione **5**

3 Package Diagram **6**
3.1 Analisi dei package 6
3.2 Versione Completa 7

4 Sequence Diagrams **8**
4.1 Crea Corsa 8
4.2 Aggiungi Natante 9

1 Dominio del problema

Questo diagramma rappresenta il dominio del problema e descrive le classi, le loro interazioni e le responsabilità identificate

1.1 Analisi dei requisiti

In questa sezione analizzeremo il dominio del problema con lo scopo di definire le funzionalità chiave che il programma deve soddisfare. Di seguito si identificano le entità, le relazioni e le regole principali richieste.

> *"Si sviluppi un sistema informativo, composto da una base di dati relazionale e da un applicativo Java dotato di GUI (Swing o JavaFX) che consenta ad un viaggiatore di conoscere e scegliere i viaggi utili per raggiungere le isole.*
>
> *Il sistema si basa sulla conoscenza delle corse offerte dalle compagnie di navigazione. Ogni corsa è offerta da una specifica compagnia di navigazione, che indica il tipo di natante utilizzato. Tra i tipi di natante si distinguono i traghetti (che trasportano persone e automezzi), gli aliscafi e le motonavi (che trasportano entrambe solo passeggeri). Ogni corsa ha cadenza giornaliera, un orario di partenza e un orario di arrivo ma può essere operata solo in alcuni giorni della settimana e solo in alcuni specifici periodi dell'anno*
>
> *"Ogni corsa ha diversi prezzi: un prezzo per il biglietto intero, uno per il biglietto ridotto. Inoltre può esserci un sovraprezzo per la prenotazione e uno per i bagagli. Ogni corsa è caratterizzata da un porto di arrivo e da uno di partenza: nel caso di corse che abbiano uno scalo intermedio il sistema espone tra le sue corse tutte le singole tratte.*
>
> *Ogni compagnia di navigazione ha un nome e una serie di contatti (telefono, mail, sito web indirizzi su diversi social).*
>
> *Il sistema può essere utilizzato dalle compagnie e dai passeggeri."*

il programma deve permettere all'utente di scegliere la tratta che desidera prenotare, visualizzando una GUI collegata a un database comprendente tutte le tratte esistenti.
Inoltre cominciamo a descrivere le prime entità: **Compagnie**, con attributi `nome, telefono, mail, sito`, il **Natante** descritto dalla compagnia di navigazione, che può essere di vari tipi in base alle necessità, ovvero `traghetti, aliscafi, motonavi`. Analizzando le **Corse** possiamo identificare `orario di partenza, orario di arrivo, giorni operata`, oltre a `Porto di Partenza, Porto di Arrivo`. Per esporre le singole tratte (nel caso una corsa specifica ne possegga diverse), possiamo aggiungere un attributo `scali`, il quale, se possiede nomi al suo interno, possono essere mostrati all'Utente.
Abbiamo bisogno di creare un'altra entità, **Biglietto**, che ha prezzi diversi per `biglietto intero, biglietto ridotto, sovrapprezzo bagagli, sovrapprezzo prenotazione`.

> *"Le compagnie possono aggiornare le proprie corse oppure segnalare l'annullamento o il ritardo di una singola corsa.*
>
> *Il passeggero può consultare il tabellone delle corse, che contiene tutte le corse da un determinato porto di partenza verso un determinato porto di destinazione, da un giorno e orario di partenza indicato e per le successive 24 ore, eventualmente filtrate in base al tipo di natante scelto o in base al prezzo.*

> *Nel tabellone le corse in ritardo o cancellate dovranno comunque essere mostrato con una annotazione che riporti questo evento."*

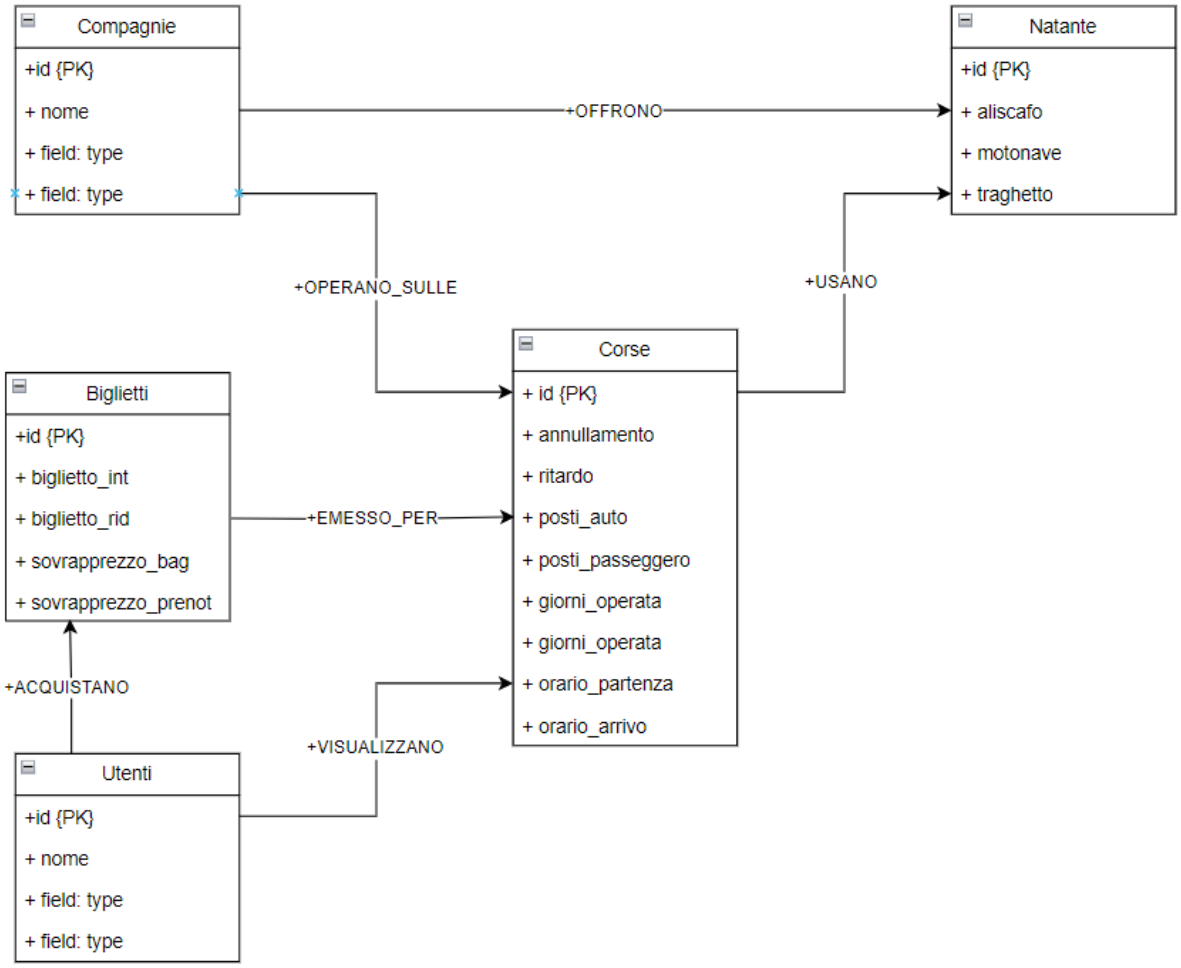
poichè il sistema sarà utilizzato sia dalle compagnie che dai passeggeri, è necessario creare un'entità **Utenti**, i quali avranno bisogno di un `nome utente` e una `password` per accedere al sistema. Inoltre, grazie a un altro attributo `usertype`, saremo capaci di distinguere gli utenti fra utenti semplici e compagnie, poiché queste ultime dovranno segnalare il `ritardo` o l'`annullamento` delle corse. Inoltre gli **Utenti** sono in grado di visualizzare le **Corse** grazie alla relazione **Utenti→Corse**, e le **Compagnie** possono modificare le corse grazie alla relazione **Compagnie→Corse**

> *"Le compagnie indicano anche la capienza dei natanti utilizzati per quella corsa, in termini di numero di posti per i passeggeri e numero di posti per autoveicoli. I passeggeri possono prenotare una singola corsa solo se il sistema verificherà la presenza di un posto utile. All'atto della prenotazione il sistema dovrà aggiornare il numero dei posti utili in tutte le tratte interessate"*

per verificare la presenza di posti utili, vanno creati due attributi `posti_auto` e `posti_passeggeri`, che indicano rispettivamente il numero di posti auto e posti per passeggeri disponibili, all'interno dell'entità **Corse**, che aggiornerà il valore delle entità ad ogni acquisto.

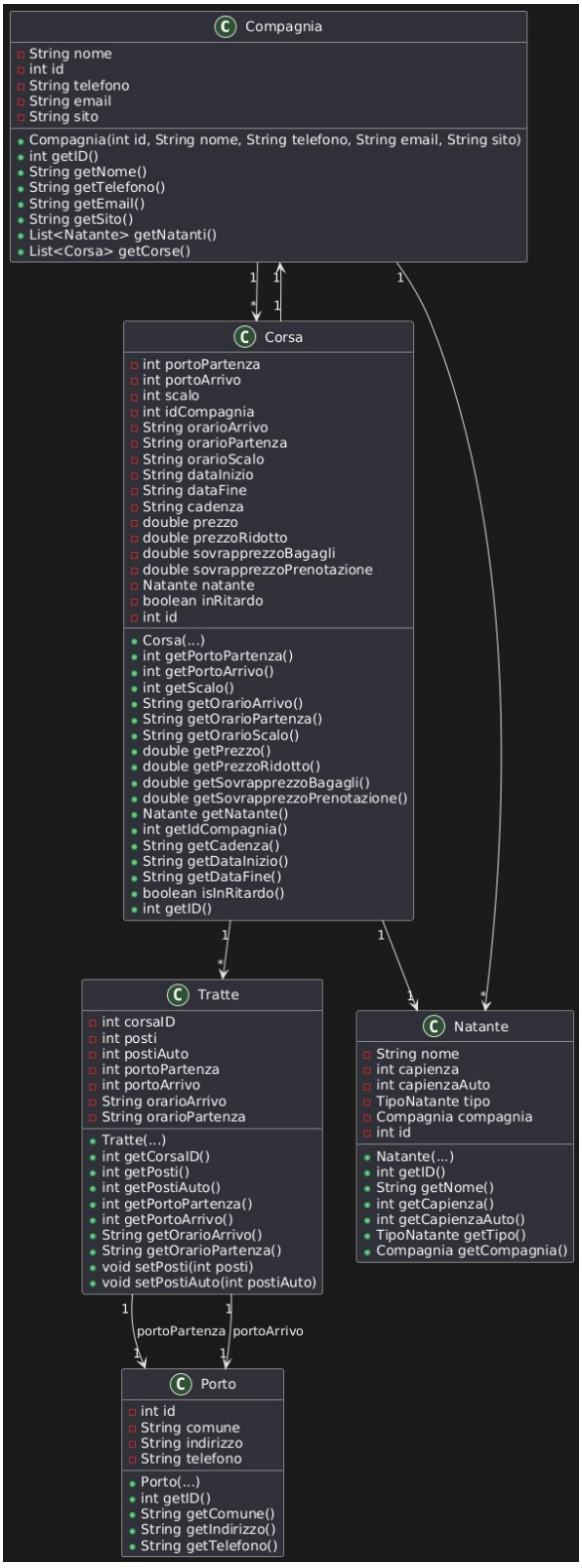
1.2 Diagramma del Dominio del Problema

Il seguente class diagram del Dominio del Problema è stato creato partendo dalle considerazioni fatte nel paragrafo precedente:



2 Dominio della Soluzione

Questo diagramma rappresenta il dominio della soluzione, fornendo una visione dettagliata del modello implementato in Java.



3 Package Diagram

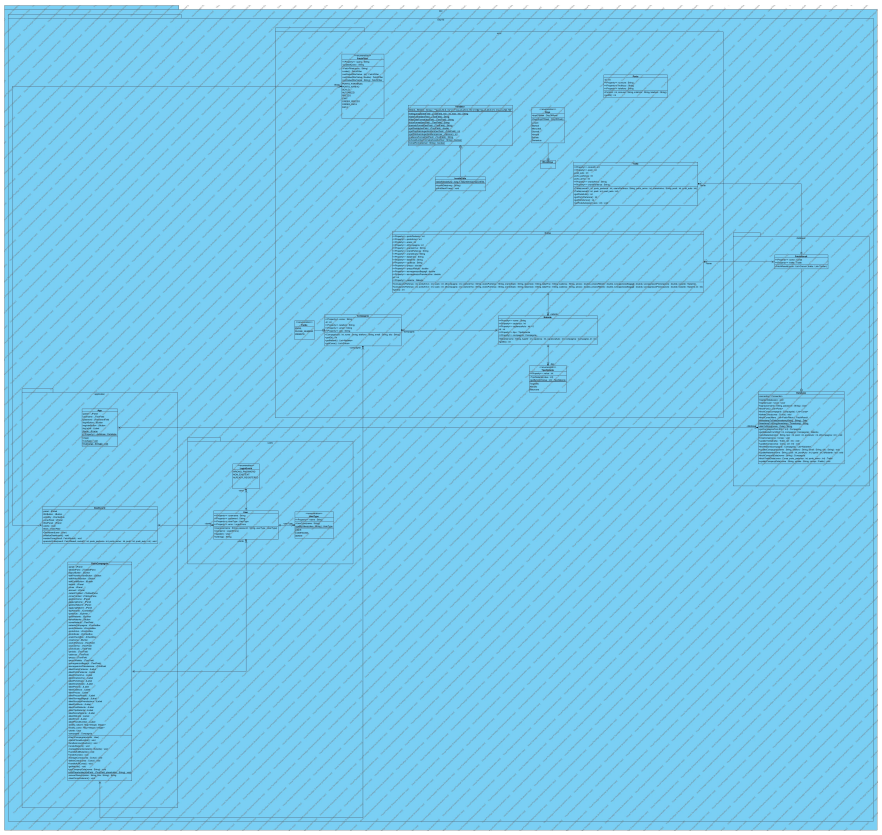
3.1 Analisi dei package

Analizziamo ora i vari package presenti, descrivendone il funzionamento.

Package	**Descrizione**
Misc	Rappresenta un sistema per gestire informazioni relative a compagnie di navigazione, natanti, corse e porti, con funzionalità di validazione per garantire l'integrità dei dati .
Users	Gestisce la registrazione e l'autenticazione degli utenti, fornendo un sistema per rappresentare gli errori di login e i diversi tipi di utenti nel sistema.
Application	Forma un sistema che gestisce l'interfaccia utente dell'applicazione, consentendo agli utenti e alle compagnie di navigazione di interagire con il sistema per gestire corse, natanti e prenotazioni.
Database	Funge da interfaccia tra l'applicazione e il database, gestendo la connessione, l'autenticazione degli utenti, e le operazioni di aggiunta/modifica/rimozione per porti, corse, natanti e compagnie.

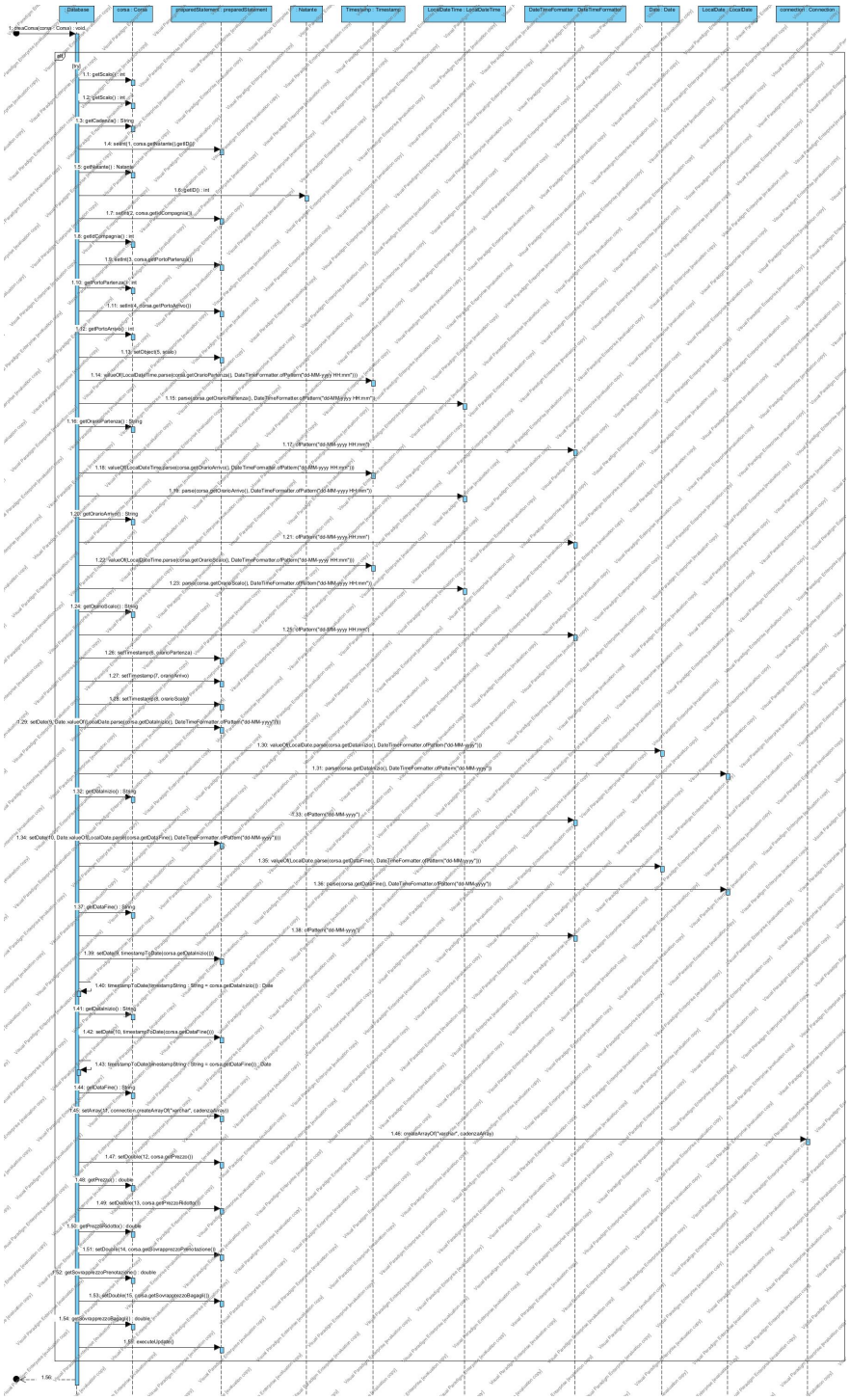
3.2 Versione Completa

Questo diagramma mostra l'architettura completa del sistema, comprendendo i package e le classi, e tutte le relazioni tra esse, creato utilizzando VisualParadigm.



4 Sequence Diagrams

4.1 Crea Corsa



4.2 Aggiungi Natante