**Università degli Studi di Salerno**

**Corso di Ingegneria del Software**

# 

**AirDreams ODD**

# 

# 

# 

# 

# 

# 

# 

**Partecipanti**

|  |  |
| --- | --- |
| **Nome** | **Matricola** |
| Noemi Cipriano | 0512105472 |
| Teresa Elia | 0512105106 |
| Rosaria Rossi | 0512105658 |
| Grazia Varone | 0512105490 |

**Indice**

1. **Introduzione……………………………………………………………………………………….….4**
   1. Object design trade-offs……………………………………………………………………....4
   2. Linee Guida per la Documentazione delle Interfacce…………………………..…5
2. **Packages………………………………………………………………………………………..……….6**
   1. Packages AirDreams……………………………………………………………………………6
      1. Package PresentationUtente……………………………………………………6
      2. Package PresentationVolo……………………………………………………….8
      3. Package PresentationCarrello………………………………………………….9
      4. Package PresentationOrdine………………………………………………….10
      5. Package PresentationCompagniaAerea…………………………………..11
   2. Package Bean……………………………………………………………………………………12
   3. Package Model………………………………………………………………………………….13
   4. Class Interfaces………………………………………………………………………………...14
      1. UtenteManager……………………………………………………………………..14
      2. BigliettoManager…………………………………………………………………..15
      3. VoloManager………………………………………………………………………...16
      4. PoliticaBagagliManager…………………………………………………………17
      5. OrdineManager……………………………………………………………………..17
      6. BagaglioManager…………………………………………………………………..18
      7. CarrelloManager…………………………………………………………………...18
      8. CartaDiCreditoManager…………………………………………………………19
      9. CompagniaAereaManager……………………………………………………...20
      10. AeroportoManager………………………………………………………………..21

## 

## **1. Introduzione**

Dopo la realizzazione dei documenti RAD e SDD abbiamo descritto il sistema e gli obiettivi, tralasciando gli aspetti implementativi. Il seguente documento ha lo scopo di produrre un modello capace di integrare le funzionalità individuate nelle fasi precedenti. In particolare, definisce le interfacce delle classi, le operazioni, i tipi, gli argomenti e le signature dei sottosistemi definiti nel System Design. Inoltre, sono specificati i trade-off e le linee guida.

## **1.1 Object design trade-offs**

**Portata vs. Latenza :**

Il sistema utilizza un flusso guidato dagli eventi. Il Web server assegna un nuovo thread per ogni richiesta, consentendo così la gestione parallela delle richieste. Con un numero elevato di richieste, quindi, questo aumenta la produttività ma anche la latenza, rallentando complessivamente il sistema.

**Spazio di memoria vs. Tempo di risposta :**

Il sistema si propone di garantire risposte alle richieste degli utenti nel minor tempo possibile in relazione al carico del sistema; ciò comporta l’inevitabile incremento di spazio di memoria necessario, dovuto al bisogno di tener traccia di una mole di dati maggiore, accessibile il più velocemente possibile.

**Funzionalità vs. Usabilità :**

Il software incorpora funzionalità che soddisfano le specifiche dei requisiti. Con MySQL per gestire i dati persistenti, Java come linguaggio di programmazione, e JSP per strutturare le pagine web l'usabilità del sistema è pienamente raggiunta.

**Rapido sviluppo vs. Funzionalità**

Seguendo un modello per lo sviluppo di sistemi software, il processo ha incluso oltre la necessaria fase di implementazione, anche quella di analisi dei requisiti, progettazione e testing. Pertanto il sistema risultante è capace di soddisfare le specifiche funzionali promesse con un preciso approccio di sviluppo delle applicazioni.

**Costo vs. riutilizzabilità**

Le interfacce software e le componenti possono essere riutilizzate all'interno del sistema per implementare funzionalità per diversi moduli; tuttavia il software è stato progettato utilizzando Java per la logica e MySQL per l'accesso ai dati e pertanto l’uso di quest’ultimo potrebbe risultare costoso.

## **1.2 Linee Guida per la Documentazione delle Interfacce**

Gli sviluppatori dovranno seguire le seguenti convenzioni per la scrittura del codice:

**Naming Convention**

* È buona norma utilizzare nomi:
  + Descrittivi
  + Pronunciabili
  + Di uso comune
  + Lunghezza medio-corta

**Variabili**

* I nomi delle variabili devono cominciare con una lettera minuscola. Se il nome della variabile è costituito da più parole, solo l’iniziale delle altre parole sarà maiuscola (notazione CamelCase), inoltre è possibile abbreviare il nome della variabile solo se non peggiora la leggibilità e comprensibilità.
* Le costanti dovranno essere scritte interamente in maiuscolo e se il nome è costituito da più parole vengono separate con l’underscore.

**Metodi**

* I nomi dei metodi devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola (notazione CamelCase). Il nome del metodo tipicamente consiste di un verbo che identifica un’azione, seguito dal nome di un oggetto.
* I nomi dei metodi per l’accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile() e setNomeVariabile().
* La descrizione dei metodi deve apparire prima di ogni dichiarazione di metodo, e deve descriverne lo scopo. Deve includere anche informazioni sugli argomenti, sul valore di ritorno, e se applicabile, sulle eccezioni.

**Classi**

I nomi delle classi devono iniziare con una lettera maiuscola.

Il nome deve fornire informazioni utili relative al loro scopo.

Ogni file sorgente contiene una singola classe e deve essere strutturato in un determinato modo:

* L’ istruzione package che permette di inserire la classe in un determinato package
* L’ istruzione import che importa le librerie necessarie alla class
* Una piccola descrizione della classe

## **1.3 Definizioni, acronimi e abbreviazioni acronimi**

Il documento e il sistema riporteranno i seguenti acronimi per l’identificazione di strumenti e/o elementi usati durante lo sviluppo.

* RAD: Requirements Analysis Document
* SDD: System Design Document
* ODD: Object Design Document
* DBMS: Database Management System
* API: Application Programming Interface
* JDBC: Java Database Connectivity
* CamelCase: pratica che consiste nello scrivere parole composte unendo tutte le parole tra loro, ma lasciando le loro iniziali maiuscole.

## **2. Packages** Il sistema è diviso in 7 packages:

* + Interfaccia utente
  + Gestione utente
  + Gestione volo
  + Gestione carrello
  + Gestione ordine
  + Gestione compagnia aerea
  + Data Access

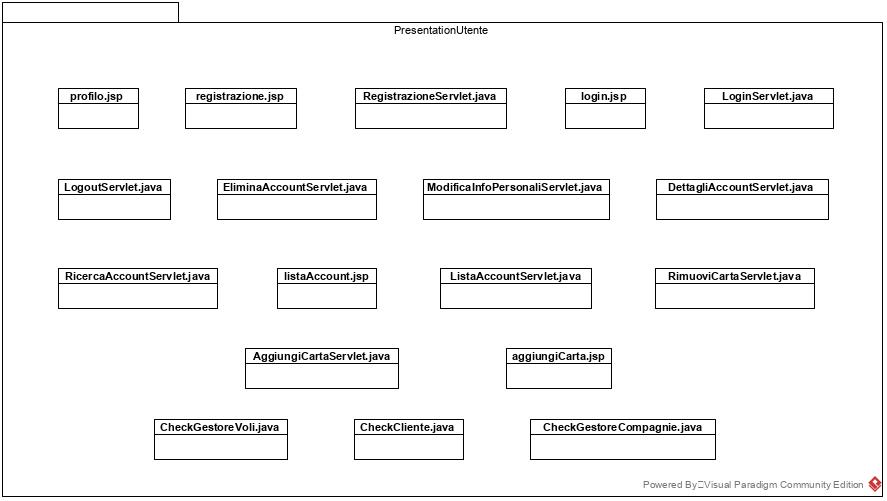
I package del sistema contengono sia classi atte ad implementare le funzionalità del sistema offerte all’utente, sia atte a gestire la logica, sia atte ad offrire risposte sotto forma di interfaccia grafica.

**Interfaccia utente**: rappresenta l’interfaccia del sistema, ed offre la possibilità all’utente di interagire con quest’ultimo, offrendo sia la possibilità di inviare, in input, che di visualizzare, in output, dati.

**Gestione utente, gestione volo, gestione carrello, gestione ordine, gestione compagnia aerea**: Si occupano dell’elaborazione dei dati da inviare al client.

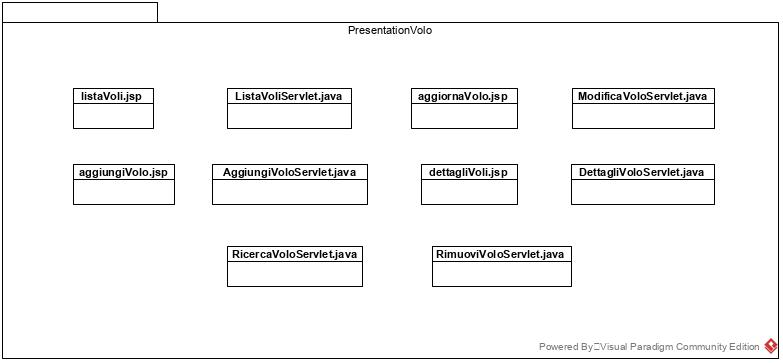
**Data access**: Ha il compito di memorizzare i dati sensibili del sistema, utilizzando un DBMS che gestisce i dati. La comunicazione verso il DBMS è realizzata tramite le API JDBC.

## **2.1 Packages AirDreams**

**2.1.1 Package PresentationUtente**  


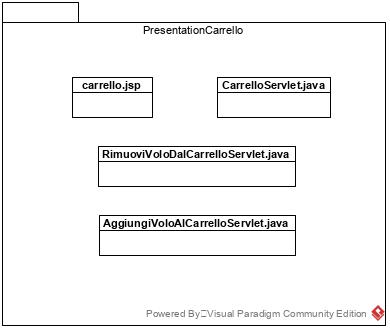
|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| profilo.jsp | Pagina che rappresenta l’area personale dell’utente. |
| aggiungiCarta.jsp | Pagina che mostra un form per aggiungere una carta di credito. |
| login.jsp | Pagina che mostra un form per effetturare l’accesso al sistema. |
| registrazione.jsp | Pagina che mostra un form per registrarsi al sistema e quindi avere un account. |
| listaAccount.jsp | Pagina che mostra la lista di tutti gli account registrati al sistema. |
| AggiungiCartaServlet | Questa servlet permette di aggiungere una nuova carta di credito relativa ad un utente. |
| ModificaInfoPersonaliServlet | Questa servlet permette di modificare le informazioni personali di un determinato account. |
| EliminaAccountServlet | Questa servlet permette di eliminare l’account di un utente. |
| RimuoviCartaServlet | Questa servlet permette di rimuovere la carta di credito di un utente. |
| LoginServlet | Questa servlet permette di far effettuare l’accesso al sistema ad un determinato utente. |
| RegistrazioneServlet | Questa servlet permette di effettuare la registrazione al sistema da parte di un utente |
| LogoutServlet | Questa servlet permette di far disconnettere un utente dal proprio account. |
| ListaAccountServlet | Questa servlet permette di visualizzare tutti gli account registrati al sistema. |
| RicercaAccountServlet | Questa servlet permette di ricercare determinati account. |
| DettagliAccountServlet | Questa servlet permette di visualizzare i dettagli di un determinato account. |
| CheckGestoreVoli | Questo filtro si occupa di assicurare che solo un gestore voli può accedere alle funzionalità a lui dedicate, come aggiungere un volo ecc. |
| CheckGestoreCompagnie | Questo filtro si occupa di assicurare che solo un gestore compagnie può accedere alle funzionalità a lui dedicate, come aggiungere una compagnia aerea ecc. |
| CheckCliente | Questo filtro si occupa di assicurare che solo un cliente può accedere alle funzionalità a lui dedicate, come aggiungere un volo al carrello, concludere un ordine ecc. |

**2.1.2 Package PresentationVolo**

****

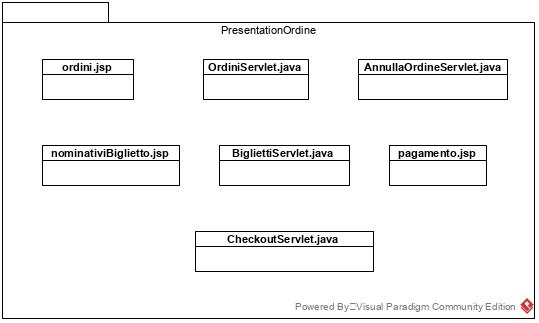
|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| listaVoli.jsp | Pagina che mostra una lista di diversi voli con le informazioni principali. |
| dettagliVolo.jsp | Pagina che mostra i precisi dettagli di un singolo volo. |
| aggiungiVolo.jsp | Pagina che mostra un form che permette di inserire informazioni riguardo un volo da aggiungere al catalogo del sistema. |
| aggiornaVolo.jsp | Pagina che mostra un form che permette di aggiornare informazioni riguardo un volo esistente. |
| ListaVoliServlet | Questa servlet permette di visualizzare una certa lista di voli. |
| AggiungiVoloServlet | Questa servlet permette di aggiungere un volo al catalogo del sistema. |
| RimuoviVoloServlet | Questa servlet permette di rimuovere un volo dal catalogo del sistema. |
| ModificaVoloServlet | Questa servlet permette di aggiornare un volo presente nel catalogo del sistema. |
| DettagliVoloServlet | Quest servlet permette di visualizzare i dettagli di un volo presente nel catalogo del sistema. |
| RicercaVoloServlet | Questa servlet permette di ricercare un volo all’interno del catalogo. |

**2.1.3 Package PresentationCarrello**

****

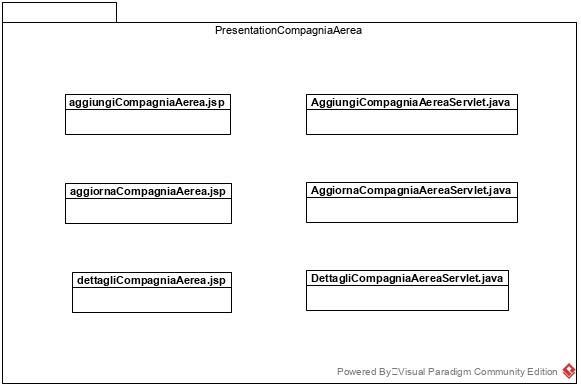
|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| carrello.jsp | Pagina che mostra il carrello di un utente con gli eventuali voli inseriti all’interno del carrello. |
| CarrelloServlet | Questa servlet permette di visualizzare il carrello relativo ad un utente. |
| AggiungiVoloAlCarrelloServlet | Questa servlet permette di aggiungere un volo al carrello relativo ad un utente. |
| RimuoviVoloDalCarrelloServlet | Questa servlet permette di rimuovere un volo dal carrello relativo ad un utente. |

**2.1.4 Package PresentationOrdine**

****

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| pagamento.jsp | Pagina che permette di selezionare una carta di credito già registrata, o di inserirne una al momento per concludere un ordine. |
| ordini.jsp | Pagina che mostra gli ordini effettuati da un determinato utente. |
| nominativiBiglietto.jsp | Pagina che mostra, durante la procedura di un ordine, il form per inserire i dati personali dei passeggeri ed aggiungere eventuali bagagli scelti. |
| OrdiniServlet | Questa servlet permette di visualizzare tutti gli ordini effettuati da un determinato utente. |
| AnnullaOrdineServlet | Questa servlet permette di annullare un ordine effettuato da un utente. |
| BigliettiServlet | Questa servlet permette di creare biglietto/i durante la procedura di ordine. |
| CheckoutServlet | Questa servlet permette di concludere l’ordine, attraverso il pagamento. |

**2.1.5 Package PresentationCompagniaAerea**

****

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| aggiungiCompagniaAerea.jsp | Pagina che mostra un form per aggiungere una nuova compagnia aerea al sistema. |
| aggiornaCompagniaAerea.jsp | Pagina che mostra un form per aggiornare i dettagli di una determinata compagnia aerea. |
| dettagliCompagniaAerea.jsp | Pagina che mostra i precisi dettagli di una compagnia aerea. |
| AggiungiCompagniaAereaServlet | Questa servlet permette di aggiungere una nuova compagnia aerea all’interno del sistema. |
| AggiornaCompagniaAereaServlet | Questa servlet permette di aggiornare i dettagli di una compagnia aerea esistente all’interno del sistema. |
| DettagliCompagniaAereaServlet | Questa servlet permette di visualizzare i dettagli di una compagnia aerea esistente all’interno del sistema. |

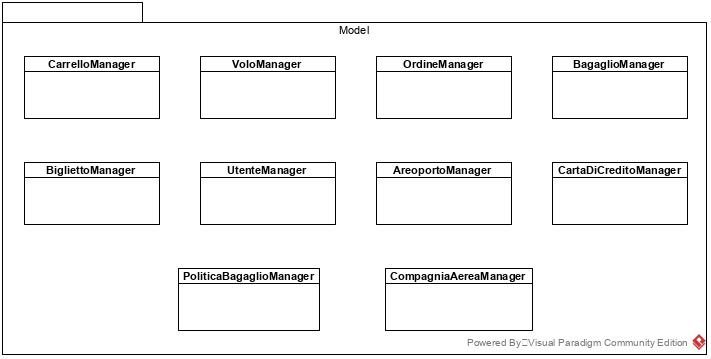
**2.2 Package Bean**

**Immagine che contiene screenshot

Descrizione generata automaticamente**

|  |  |
| --- | --- |
| Classe | Descrizione |
| Ordine | Questa classe rappresenta l’ordine concluso da un cliente. |
| Carrello | Questa classe rappresenta il carrello di ogni cliente. |
| Volo | Questa classe contiene tutte le informazioni di un possibile volo offerto dal sistema. |
| Aeroporto | Questa classe rappresenta l’aeroporto, ovvero un luogo dove sono installati tutti gli impianti necessari al decollo, atterraggio e allo stazionamento degli aerei. |
| CartaDiCredito | Questa classe rappresenta una carta di credito memorizzata per un cliente e che può essere utilizzata per l’acquisto di biglietti aerei. |
| Biglietto | Questa classe rappresenta il biglietto per un determinato volo con le sue informazioni. |
| CompagniaAerea | Questa classe tiene traccia della compagnia aerea con le relative informazioni. |
| BagaglioMano | Questa classe rappresenta il bagaglio a mano incluso nel biglietto di un singolo passeggero con le relative dimensioni. |
| BagaglioStiva | Questa classe rappresenta il bagaglio a stiva incluso nel biglietto di un singolo passeggero con le relative dimensioni e prezzo. |
| PoliticaBagaglioStiva | Questa classe definisce la politica sul bagaglio a stiva di una singola compagnia aerea. |
| PoliticaBagaglioMano | Questa classe definisce la politica sul bagaglio a mano di una singola compagnia aerea. |
| Account | Questa classe rappresenta l’utente registrato, che è un qualsiasi utente che si è registrato al sistema e può effettuare il login in qualunque momento, accedendo alla propria area personale. |

**2.3 Package Model**

****

|  |  |
| --- | --- |
| CarrelloManager | Questa classe si occupa di implementare i metodi per effettuare l’inserimento, la cancellazione e la ricerca di un carrello relativo ad un particolare utente, memorizzato all’interno del DB. |
| VoloManager | Questa classe si occupa di implementare i metodi per effettuare l’inserimento, la cancellazione e la ricerca di un volo presente nel catalogo di sistema, memorizzato all’interno del DB. |
| OrdineManager | Questa classe si occupa di implementare i metodi per effettuare l’inserimento, la cancellazione e la ricerca di un ordine relativo ad un dato utente che lo ha eseguito, memorizzato all’interno del DB. |
| BagaglioManager | Questa classe si occupa di implementare i metodi per effettuare l’inserimento, la cancellazione e la ricerca di un bagaglio relativo ad un dato biglietto presente in un ordine di un particolare cliente, memorizzato all’interno del DB. |
| BigliettoManager | Questa classe si occupa di implementare i metodi per effettuare l’inserimento, la cancellazione e la ricerca di un biglietto relativo ad un dato ordine di un cliente, memorizzato all’interno del DB. |
| UtenteManager | Questa classe si occupa di implementare i metodi per effettuare l’inserimento, la cancellazione, la modifica e la ricerca di un utente registrato al sistema, memorizzato all’interno del DB. |
| AeroportoManager | Questa classe si occupa di implementare i metodi per effettuare l’inserimento, la cancellazione e la ricerca di un aeroporto possibilmente utilizzato in un volo come luogo di partenza o destinazione, memorizzato all’interno del DB. |
| CompagniaAereaManager | Questa classe si occupa di implementare i metodi per effettuare l’inserimento, la cancellazione e la ricerca di una compagnia aerea di cui il sistema offre i voli, memorizzata all’interno del DB. |
| PoliticaBagaglioManager | Questa classe si occupa di implementare i metodi per effettuare l’inserimento, la cancellazione e la ricerca di una politica bagaglio relativa ad una data compagnia aerea, memorizzata all’interno del DB. |
| CartaDiCreditoManager | Questa classe si occupa di implementare i metodi per effettuare l’inserimento, la cancellazione e la ricerca di una carta di credito relativa ad un dato utente, memorizzata all’interno del DB. |

**2.4 Class interfaces**

**2.4.1 UtenteManager**

|  |  |
| --- | --- |
| **UtenteManager** | |
| **signUp** | **Context:** UtenteManager.signUp(utente)  **Pre:** utente!=null &&utente.email!=null && utente.nome!=null && utente.cognome!=null && utente.password!=null &&  getListaUtenti()->forAll(u |u.email!=utente.email);  **Post:** getListaUtenti()->include(utente) |
| **signIn** | **Context:** UtenteManager.signIn(utente)  **Pre:** utente!=null && utente.email!=null utente.password!=null |
| **eliminaAccount** | **Context:** UtenteManager.eliminaAccount(email)  **Pre:** getListaUtenti()->exists(u |u.email==email)  **Post**: !getListaUtenti()->exists(u |u.email==email) |
| **visualizzaInfoUtente** | **Context:** UtenteManager.visualizzaInfoUtente(email)  **Pre:** getListaUtenti()->exists(u |u.email==email)  **Post:** getListaUtenti()->select(u |u.email==email) |
| **aggiornaProfilo** | **Context:** UtenteDAO.aggiornaProfilo(utenteVecchio, utenteNuovo)  **Pre:**utenteNuovo!=null &&utenteNuovo.email!=null && utenteNuovo.nome!=null && utenteNuovo.cognome!=null && utenteNuovo.password!=null &&  getListaUtenti().reject(u|u.emai==utenteVecchio.email)->forAll(u|u.email!=utenteNuovo.email)  &&  getListaUtenti()->exists(u|u.email==utenteVecchio.email)  **Post:**  getListaUtenti()->include(u|u.email==utenteNuovo.email) |

**2.4.2 BigliettoManager**

|  |  |
| --- | --- |
| **BigliettoManager** | |
| **doCreate** | **Context:** BigliettoManager.doCreate(biglietto)  **Pre:** biglietto.nome!=null && biglietto.cognome!=null &&  biglietto.sesso!=null &&  biglietto.codBiglietto!=null &&  biglietto.prezzoBiglietto!=null &&  biglietto.ordine!=null &&  biglietto.bagaglio!=null  **Post:** getListaBiglietti()->include(biglietto) |
| **doDelete** | **Context:** BigliettoManager.doDelete(codBiglietto)  **Pre:** getListaBiglietti()->exists(b|b.codBiglietto == codBiglietto)  **Post:** !getListaBiglietti()->exists(b|b.codBiglietto == codBiglietto) |
| **dettagliBiglietto** | **Context:** BigliettoManager.dettagliBiglietto(codBiglietto)  **Pre:** getListaBiglietti()i->exists(b|b.codBiglietto == codBiglietto)  **Post:** getListaBiglietti()->select(b|b.codBiglietto == codBiglietto) |

**2.4.3 VoloManager**

|  |  |
| --- | --- |
| **VoloManager** | |
| **inserisciVolo** | **Context:** VoloManager..inserisciVolo(volo)  **Pre:** volo.idVolo != null &&  volo.dataPart != null &&  volo.prezzo != null &&  volo.postiDisponibili != null &&  volo.durata != null &&  volo.orarioPart != null &&  volo.bagaglioStivaCompreso != null  **Post:** getListaVoli()->include(volo) |
| **rimuoviVolo** | **Context:** VoloManager.rimuoviVolo(idVolo)  **Pre:** getListaVoli()->exists(v|v.idVolo==idVolo)  **Post:** !getListaVoli()->exists(v|v.idVolo==idVolo) |
| **aggiornaVolo** | **Context:** VoloManager.aggiornaVolo(volo)  **Pre:** volo!=null &&  getListaVoli()->exists(v|v.idVolo==volo.idVolo)  **Post:** getListaVoli()->include(volo) |
| **dettaglioVolo** | **Context:** VoloManager.dettagliVolo(idVolo)  **Pre:** idVolo!=null && getListaVoli()->exists(v|v.idVolo==idVolo)  **Post:** getListaVoli()->select(v|v.idVolo == idVolo) |

**2.4.4 PoliticaBagagliManager**

|  |  |
| --- | --- |
| **PoliticaBagagliManager** | |
| **insertBagaglio** | **Context:** PoliticaBagagliManager.insertBagaglio(polbagaglio)  **Pre:** polbagaglio.peso!=null && polbagaglio.dimensioni!=null && polbagaglio.compagniaAerea!=null  **Post:** getPolitiche()->include(polbagaglio) |
| **updateBagaglio** | **Context:** PoliticaBagaglioManager.updateBagaglio(polbagaglio)  **Pre:** getPolitiche()-> exists(p|p.compagniaAerea == polBagaglio.compagniaAerea)  **Post:** getPolitiche()-> include(p|p.compagniaAerea == polBagaglio.compagniaAerea) |
| **updateBagaglioAStiva** | **Context:** PoliticaBagaglioManager.updateBagaglioAStiva(polbagaglio)  **Pre:** getPolitiche()-> exists(p|p.compagniaAerea == polBagaglio.compagniaAerea)  **Post:** getPolitiche()-> include(p|p.compagniaAerea == polBagaglio.compagniaAerea) |
| **dettagliPoliticaBagagli** | **Context:** PoliticaBagaglioManager.dettagliPoliticaBagagli(compagniaAerea)  **Pre:** getPolitiche()-> exists((p|p.compagniaAerea == compagniaAerea )&& getPolitiche()-> exists((p|p.compagniaAerea == compagniaAerea)  **Post:** getPolitiche().> select((p|p.compagniaAerea == compagniaAerea) && getPolitiche()->select(p|p.compagniaAerea == compagniaAerea) |

**2.4.5 OrdineManager**

|  |  |
| --- | --- |
| **OrdineManager** | |
| **creaOrdine** | **Context:** ordineManager.creaOrdine(ordine)  **Pre:** ordine!=null && ordine.codOrdine =! null && ordine.dataAcquisto !=null && ordine.carta!=null && ordine.biglietti!=null  **Post:** getOrdini()->include(ordine) &&  ordine.biglietti->  forAll(b|b.volo.getPostiDisponibili()==  b.volo.@pre.getPostiDisponibili()-1) |
| **annullaOrdine** | **Context**: OrdineManager.annullaOrdine(codOrdine)  **Pre**: codOrdine!= null &&  getOrdini()->exist(o|o.codOrdine == codOrdine)  &&  ordine.biglietti->forAll(b | LocalDate.now()-b.volo.getDataPartenza() >= Calendar.WEEK)  **Post**: !getOrdini()->select(o|o.codOrdine == codOrdine) &&  ordine.biglietti->  forAll(b|b.volo.getPostiDisponibili()==  b.volo.@pre.getPostiDisponibili()+1) |
| **visualizzaDettagliOrdine** | **Context**: OrdineManager.visualizzaDettagliOrdine(codOrdine)  **Pre**: codOrdine != null && getOrdini()-> exist(o|o.codOrdine == codOrdine)  **Post**: getOrdini()->select(o|o.codOrdine == codOrdine); |

## **2.4.6 BagaglioManager**

|  |  |
| --- | --- |
| **BagaglioManager** | |
| **inserisciBagaglio** | **Context:** bagaglioManager.inserisciBagaglio(bagaglio)  **Pre:** bagaglio!=null && bagaglio.peso =! null && bagaglio.dimensioni !=null  **Post:** getBagagli()->include(bagaglio) |

## **2.4.7 CarrelloManager**

|  |  |
| --- | --- |
| **CarrelloManager** | |
| **visualizzaDettagliCarrello** | **Context**: CarrelloManager.visualizzaDettagliCarrello(email)  **Pre**: email != null  **Post**: getCarrelli()-> select(c|c.email == email); |
| **aggiungiVoloAlCarrello** | **Context:** CarrelloManager.aggiungiVoloAlCarrello(email, volo)  **Pre**: volo != null && getCarrelli()-> exists(c|c.email == email)  **Post**: (getCarrelli()->select(c|c.email==email).getVoli().size==(getCarrelli()->select(c|c.email==email).getVoli().size+1 |
| **rimuoviVoloDalCarrello** | **Context**: CarrelloManager.rimuoviVoloDalCarrello(email, idVolo)  **Pre**: getCarrelli()-> exists(v|v.email == email) && id Volo!=null  **Post:** (getCarrelli()>select(c|c.email==email).getVoli().size==(getCarrelli()->select(c|c.email==email).getVoli().size-1 |

**2.4.8 CartaDiCreditoManager**

|  |  |
| --- | --- |
| **CartaDiCreditoManager** | |
| **creaCartaDiCredito** | **Context:** cartaDiCreditoManager.creaCartaDiCredito(cartaDiCredito)  **Pre:** cartaDiCredito!=null && cartaDi cartaDiCredito.email =! null && Credito.nCarta =! null && cartaDiCredito.titolare !=null && cartaDiCredito.dataScadenza!=null && cartaDiCredito.cvc!=null  **Post:** getCarte()>include(cartaDiCredito) |
| **eliminaCartaDiCredito** | **Context:** cartaDiCreditoManager.eliminaCartaDiCredito(ncarta)  **Pre:** getCarte()-> exists(c|c.nCarta == ncarta)  **Post:** !getCarte() -> exists(c|c.nCarta == ncarta) |
| **visualizzaInfoCartaDiCredito** | **Context:** cartaDiCreditoDAO.visualizzaInfoCartaDiCredito(ncarta)  **Pre:** getCarte()-> exists(c|c.nCarta == ncarta)  **Post:** getCarte -> select(c|c.nCarta == ncarta) |

**2.4.9 CompagniaAereaManager**

|  |  |
| --- | --- |
| **CompagniaAereaManager** | |
| **aggiungiCompagniaAerea** | **Context:** CompagniaAreaManager.aggiungiCompagniaArea(compagniaArea)  **Pre:** compagniaArea!=null && compagniaAerea.nome =! null && compagniaAerea.sito !=null &&  **Post:** getCompagnieAeree()->include(compagniaAerea) |
| **rimuoviCompagniaArea** | **Context:** compagniaAereaManager.rimuoviCompagniaAerea(nome)  **Pre:** getCompagnieAeree-> exists(c|c.nome == nome)  **Post:** !getCompagnieAeree() -> select(c|c.nome == nome) |
| **visualizzaInfoCompagniaArea** | **Context:** compagniaAereaManager.visualizzaInfoCompagniaAerea(nome)  **Pre:** getCompagnieAeree()-> exists(c|c.nome == nome)  **Post:** getCompagnieAeree() -> select(c|c.nome == nome) |

**2.4.10 AeroportoManager**

|  |  |
| --- | --- |
| **AeroportoManager** | |
| **aggiungiAeroporto** | **Context:** AeroportoManager.aggiungiAeroporto(aeroporto)  **Pre:** aeroporto !=null && aeroporto.codice =! null && aeroporto.nome !=null &&  aeroporto.città !=null &&  aeroporto.stato !=null  **Post:** getAeroporti()>include(aeroporto) |
| **trovaVoloPerNome** | **Context:** AeroportoManager.trovaVoloPerNome(stringa)  **Pre:** stringa !=null |

## 