



Università degli Studi di Salerno  
Corso di Ingegneria del Software



BookPoint  
ODD  
Versione 1.0

Partecipanti:

Nome Cognome	Matricola
Massimo Giordano	0512104480
Giovanni Buonocore	0512104612
Gennaro Teodoro	0512104876

## SOMMARIO

---

1	Introduzione .....	3
---	--------------------	---

1.1	Object Design Trade-offs .....	3
1.2	Linee Guida per la documentazione delle interfacce .....	3
1.3	Definizioni, acronimi e abbreviazioni .....	4
1.4	Riferimenti .....	4
2	Packages .....	5
2.1	control .....	5
2.1.1	gestioneAccount .....	5
2.1.2	gestioneAmministratore .....	6
2.1.3	gestioneAmministratoreOrdini.....	6
2.1.4	gestioneRicerca .....	6
2.1.5	gestioneOrdine .....	7
2.1.6	gestioneInterazioneLibro .....	7
2.1.7	gestioneAcquisto .....	7
2.2	model.....	8
2.3	bean.....	8
2.4	Carrello .....	9
2.5	connectionPool .....	9
2.6	presentation.....	9
2.6.1	presentation amministratore .....	9
2.6.2	presentation amministratore ordini .....	9
2.6.3	presentation cliente.....	10
3	Class interfaces .....	11
3.1	Interfacce Bean .....	11
3.2	AccountManager .....	12
3.3	AmministratoreManager .....	13
3.4	AmministratoreOrdiniManager.....	15
3.5	LibroManager .....	16
3.6	InterazioneLibroManager .....	17
3.7	GestioneOrdineManager .....	18
3.8	OrdineManager .....	19
4	Design Pattern .....	20
4.1	Object Pool Pattern .....	20

# 1 INTRODUZIONE

---

## 1.1 OBJECT DESIGN TRADE-OFFS

### **Comprensibilità vs Tempo:**

Il codice del sistema deve essere comprensibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare. Al fine di rispettare queste linee guida il codice sarà integrato da commenti volti a migliorarne la leggibilità; tuttavia questo richiederà una maggiore quantità di tempo necessario per lo sviluppo del nostro progetto.

### **Prestazioni vs Costi:**

Dal momento che il budget allocato è spendibile principalmente in risorse umane e non consente l'acquisto di tecnologie proprietarie specifiche verranno utilizzati template open source e componenti hardware di nostra proprietà.

### **Interfaccia vs Usabilità:**

Verrà realizzata un'interfaccia grafica chiara e concisa, usando form e pulsanti predefiniti che hanno lo scopo di rendere semplice l'utilizzo del sistema da parte dell'utente finale.

### **Sicurezza vs Efficienza:**

La sicurezza rappresenta uno degli aspetti principali del sistema. Tuttavia, a causa di tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su email e password.

## 1.2 LINEE GUIDA PER LA DOCUMENTAZIONE DELLE INTERFACCE

Gli sviluppatori dovranno seguire precise linee guida per la stesura del codice:

### **Naming Convention:**

Per la documentazione delle interfacce bisognerà utilizzare nomi:

- Descrittivi
- Pronunciabili;
- Di lunghezza medio-corta;
- Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

### **Variabili:**

- I nomi delle variabili dovranno iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola.
- In ogni riga dovrà esserci un'unica variabile dichiarata, eventualmente allineata con quelle del blocco dichiarativo.
- In determinati casi, è possibile utilizzare il carattere underscore “\_”: il caso principale previsto è quello relativo alla dichiarazione di costanti oppure di proprietà statiche.

### **Metodi:**

- I nomi dei metodi dovranno iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola, secondo la “Camel Notation”.

- Il nome del metodo sarà costituito da un verbo che ne identifica l'azione seguito da un sostantivo, eventualmente aggettivato.
- Il nome dei metodi accessori e modificatori seguirà, rispettivamente, i pattern `getNomeVariabile` e `setNomeVariabile`.
- Ai metodi verrà aggiunto un commento `JavaDoc`, il quale deve essere posizionata prima della dichiarazione del metodo, e deve descriverne lo scopo.

#### **Classi Java e pagine JSP:**

- I nomi delle classi e delle pagine dovrà iniziare con la lettera maiuscola, così come le parole successive all'interno del nome.
- I nomi delle classi e delle pagine dovrà corrispondere alle informazioni e le funzioni fornite da quest'ultime.
- Le classi saranno strutturate prevedendo rispettivamente:
  1. Dichiarazione della classe pubblica;
  2. Dichiarazione di costanti;
  3. Dichiarazioni di variabili di classe;
  4. Dichiarazioni di variabili di istanza;
  5. Costruttore;
  6. Metodi;

#### **Packages:**

- I nomi dei packages dovranno essere scritti in minuscolo concatenando insieme diversi sostantivi o sigle, separate dal carattere ".".
- Non saranno ammessi caratteri speciali.

### **1.3 DEFINIZIONI, ACRONIMI E ABBREVIAZIONI**

#### **Acronimi:**

- RAD: Requirements Analysis Document
- SDD: System Design Document
- ODD: Object Design Document

### **1.4 RIFERIMENTI**

Documento RAD\_BookPoint.pdf

Documento SDD\_BookPoint.pdf

## 2 PACKAGES

---

Il sistema è in diviso orizzontalmente in package come segue:

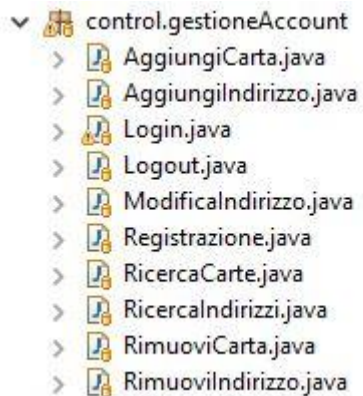
- control
- model
- bean
- carrello
- connectionPool
- presentation
- test

### 2.1 CONTROL

Il package control è suddiviso verticalmente in package come segue:

- gestioneAccount
- gestioneAmministratore
- gestioneAmministratoreOrdini
- gestioneRicerca
- gestioneOrdine
- gestioneInterazioneLibro
- gestioneAcquisto









#### 2.1.1 gestioneAccount



Classe	Descrizione
AggiungiCarta	Servlet che gestisce l'aggiunta di una carta di credito
AggiungiIndirizzo	Servlet che gestisce l'aggiunta di un nuovo indirizzo
Login	Servlet che permette di effettuare il login
Logout	Servlet che permette di effettuare il logout
ModificaIndirizzo	Servlet che gestisce la modifica di un indirizzo
Registrazione	Servlet che gestisce la registrazione di una nuovo account
RicercaCarte	Servlet che gestisce la ricerca di carte
RicercaIndirizzi	Servlet che gestisce la ricerca di indirizzi





RimuoviCarta	Servlet che gestisce la rimozione di una carta di credito
RimuoviIndirizzo	Servlet che gestisce la rimozione di un indirizzo

### 2.1.2 gestioneAmministratore

▼  control.gestioneAmministratore
>  AggiungiLibro.java
>  CambiaTipo.java
>  EliminaLibro.java
>  EliminaRecensione.java
>  EliminaUtente.java
>  ModificaAttributo.java
>  RicercaAccount.java





Classe	Descrizione
AggiungiLibro	Servlet che gestisce l'aggiunta di un libro
CambiaTipo	Servlet che gestisce il cambiamento del tipo di un libro
EliminaLibro	Servlet che gestisce l'eliminazione di un libro
EliminaRecensione	Servlet che gestisce l'eliminazione di una recensione
EliminaUtente	Servlet che gestisce l'eliminazione di un utente
ModificaAttributo	Servlet che gestisce la modifica di un attributo
RicercaAccount	Servlet che gestisce la ricerca di un account

### 2.1.3 gestioneAmministratoreOrdini

▼  control.gestioneAmministratoreOrdini
>  CambiaDataEOra.java
>  CambiaStato.java
>  RicercaOrdine.java


Classe	Descrizione
CambiaDataEOra	Servlet che gestisce il cambiamento della data e l'ora
CambiaStato	Servlet che gestisce il cambiamento dello stato di un ordine
RicercaOrdine	Servlet che gestisce la ricerca di un ordine

### 2.1.4 gestioneRicerca

▼  control.gestioneRicerca
>  Ricerca.java
>  VisualizzaCatalogo.java
>  VisualizzaLibro.java







Classe	Descrizione
Ricerca	Servlet che gestisce la ricerca di un libro
VisualizzaCatalogo	Servlet che gestisce la visualizzazione del catalogo
VisualizzaLibro	Servlet che gestisce la visualizzazione di un libro

### 2.1.5 gestioneOrdine

- ▼  control.gestioneOrdine
  - >  VisualizzaFattura.java
  - >  VisualizzaStorico.java








Classe	Descrizione
VisualizzaFattura	Servlet che gestisce la visualizzazione della fattura
VisualizzaStorico	Servlet che gestisce la visualizzazione dello storico

### 2.1.6 gestioneInterazioneLibro

- ▼  control.gestioneInterazioneLibro
  - >  AggiungiLibroPreferiti.java
  - >  AggiungiRecensione.java
  - >  RimuoviLibroPreferiti.java
  - >  RimuoviRecensione.java
  - >  VisualizzaPreferiti.java

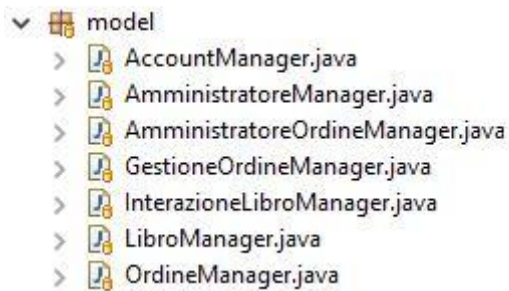
Classe	Descrizione
AggiungiLibroPreferiti	Servlet che gestisce l'aggiunta dei libri ai preferiti
AggiungiRecensione	Servlet che gestisce l'aggiunta di una recensione ad un libro
RimuoviLibroPreferiti	Servlet che gestisce l'eliminazione dei libri dai preferiti
RimuoviRecensione	Servlet che gestisce l'eliminazione di una recensione di un libro
VisualizzaPreferiti	Servlet che gestisce la visualizzazione dei libri preferiti

### 2.1.7 gestioneAcquisto

- ▼  control.gestioneAcquisto
  - >  AggiungiAlCarrello.java
  - >  AumentaQuantità.java
  - >  CompletaAcquisto.java
  - >  DiminuisciQuantità.java
  - >  EliminaDalCarrello.java
  - >  SelezionaCartaEIndirizzo.java

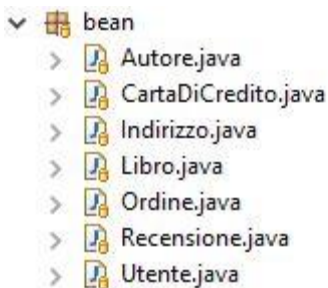
Classe	Descrizione
AggiungiAlCarrello	Servlet che gestisce l'aggiunta al carrello di un libro
AumentaQuantità	Servlet che gestisce l'aumento della quantità di un libro
CompletaAcquisto	Servlet che gestisce il completamento dell'acquisto di un libro
DiminuisciQuantità	Servlet che gestisce la diminuzione della quantità di un libro
EliminaDalCarrello	Servlet che gestisce l'eliminazione di un libro dal carrello
SelezionaCartaEIndirizzo	Servlet che gestisce la gestione di una carta e di un indirizzo

## 2.2 MODEL



Classe	Descrizione
AccountManager	Classe che implementa le operazioni riguardanti l'account
AmministratoreManager	Classe che implementa le operazioni effettuabili dall'amministratore
AmministratoreOrdineManager	Classe che implementa le operazioni effettuabili dall'amministratore degli ordini
GestioneOrdineManager	Classe che implementa le operazioni di gestione degli ordini e visualizzazione della fattura
InterazioneLibroManager	Classe che implementa le operazioni di gestione delle recensioni e gestione dei preferiti
LibroManager	Classe che implementa la ricerca e visualizzazione dei libri
OrdineManager	Classe che implementa la memorizzazione di un ordine

## 2.3 BEAN



Classe	Descrizione
Autore	Classe che rappresenta le informazioni relative ad un autore
CartaDiCredito	Classe che rappresenta le informazioni relative ad una carta di credito
Indirizzo	Classe che rappresenta le informazioni relative ad un indirizzo
Libro	Classe che rappresenta le informazioni relative ad un libro
Ordine	Classe che rappresenta le informazioni relative ad un ordine
Recensione	Classe che rappresenta le informazioni relative ad una recensione
Utente	Classe che rappresenta le informazioni relative ad un utente





## 2.4 CARRELLO

▼  carrello  
    >  Carrello.java






Classe	Descrizione
Carrello	Classe che rappresenta un carrello, con le relative operazioni

## 2.5 CONNECTIONPOOL








▼  connectionPool  
    >  DriverMaagerConnectionPool.java

Classe	Descrizione
DriverMaagerConnectionPool	Classe che implementa l'Object pool pattern, responsabile di fornire le connessioni al database.

## 2.6 PRESENTATION

▼  presentation  
    >  amministratore  
    >  amministratoreOrdini  
    >  cliente  
    >  image

### 2.6.1 presentation amministratore

▼  amministratore  
    >  Amministratore.js  
    >  AmministratoreAccount.jsp  
    >  AmministratoreCatalogo.jsp  
    >  AmministratoreVisualizzaLibro.jsp  
    >  headerAmministratore.jsp  
    >  styleAmministratore.css

Classe	Descrizione
AmministratoreAccount	Sezione in cui l'amministratore può gestire gli account, cambiando il loro tipo o eliminandoli
AmministratoreCatalogo	Sezione in cui l'amministratore può visualizzare e aggiungere i libri
AmministratoreVisualizzaLibro	Sezione in cui l'amministratore può visualizzare e modificare le informazioni di un libro

### 2.6.2 presentation amministratore ordini

▼  amministratoreOrdini  
    >  AmministratoreOrdini.js  
    >  AmministratoreOrdiniOrdine.jsp  
    >  headerAmministratoreOrdini.jsp  
    >  styleAmministratoreOrdini.css

Classe	Descrizione
--------	-------------

AmministratoreOrdiniOrdine	Sezione in cui l'amministratore può visualizzare un ordine e modificarne lo stato, la data e l'ora di consegna
----------------------------	----------------------------------------------------------------------------------------------------------------

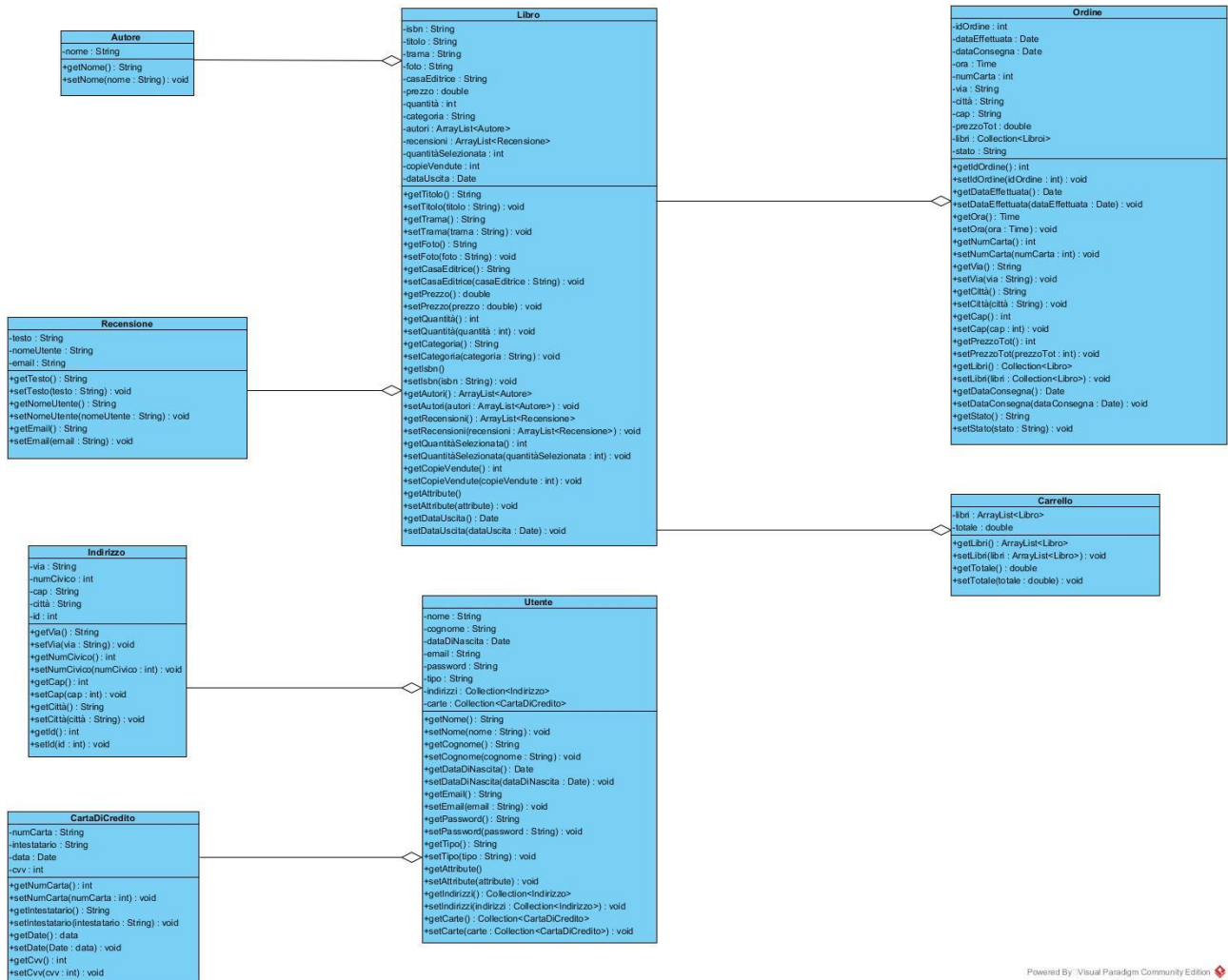
### 2.6.3 presentation cliente

▼ cliente	
AcquistoCompletato.jsp	
AreaPersonale.jsp	
Carrello.jsp	
> cliente.js	
CompletaAcquisto.jsp	
Fattura.jsp	
footer.jsp	
GestioneAccount.jsp	
GestioneCarteDiCredito.jsp	
GestioneIndirizzi.jsp	
header.jsp	
Home.jsp	
IMieiOrdini.jsp	
Libri.jsp	
Login.jsp	
Preferiti.jsp	
Registrazione.jsp	
styleCliente2.css	
VisualizzaLibro.jsp	

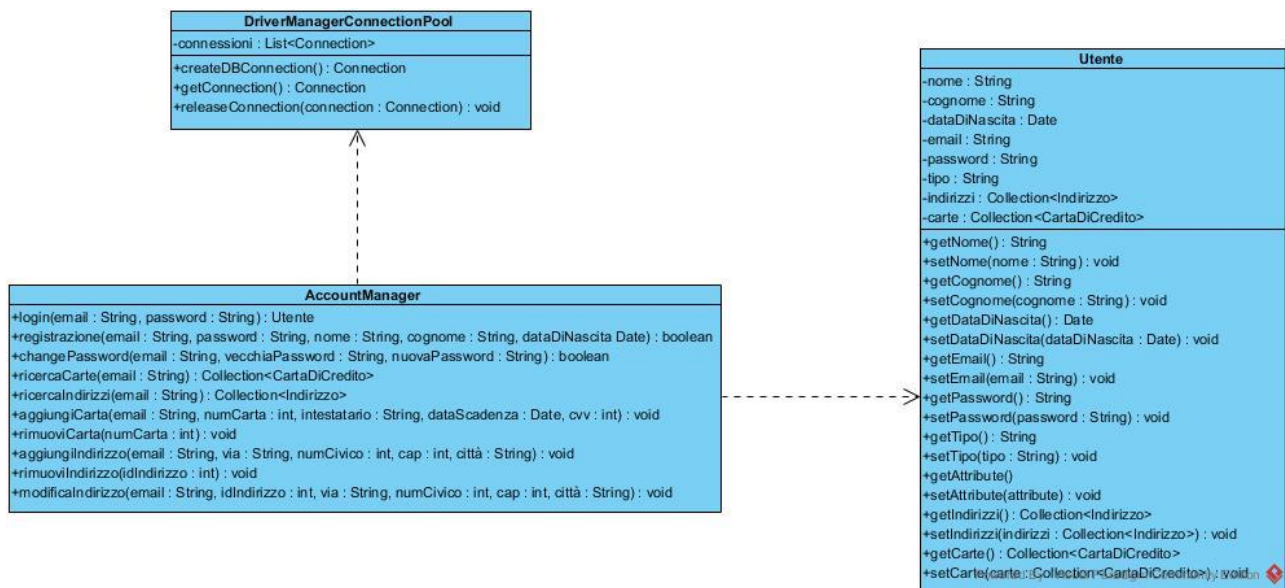
Classe	Descrizione
AcquistoCompletato	Pagina che mostra all'utente l'avvenuto completamento dell'acquisto
AreaPersonale	Sezione che mostra l'area personale di un utente registrato
Carrello	Pagina che mostra il carrello
CompletaAcquisto	Pagina che permette di completare un acquisto
Fattura	Pagina che mostra la fattura
GestioneAccount	Sezione dove l'utente può cambiare la password
GestioneCarteDiCredito	Sezione dove l'utente può visualizzare le proprie carte registrate
GestioneIndirizzi	Sezione dove l'utente può visualizzare gli indirizzi registrati
Home	Pagina iniziale del sistema
IMieiOrdini	Sezione dove l'utente può visualizzare lo storico degli ordini
Libri	Pagina che mostra i libri
Login	Pagina che permette di effettuare il login
Preferiti	Pagina che mostra i libri aggiunti ai preferiti
Registrazione	Pagina che permette di registrare un nuovo account
VisualizzaLibro	Pagina che mostra le informazioni riguardanti un libro con le relative recensioni

## 3 CLASS INTERFACES

### 3.1 INTERFACCE BEAN

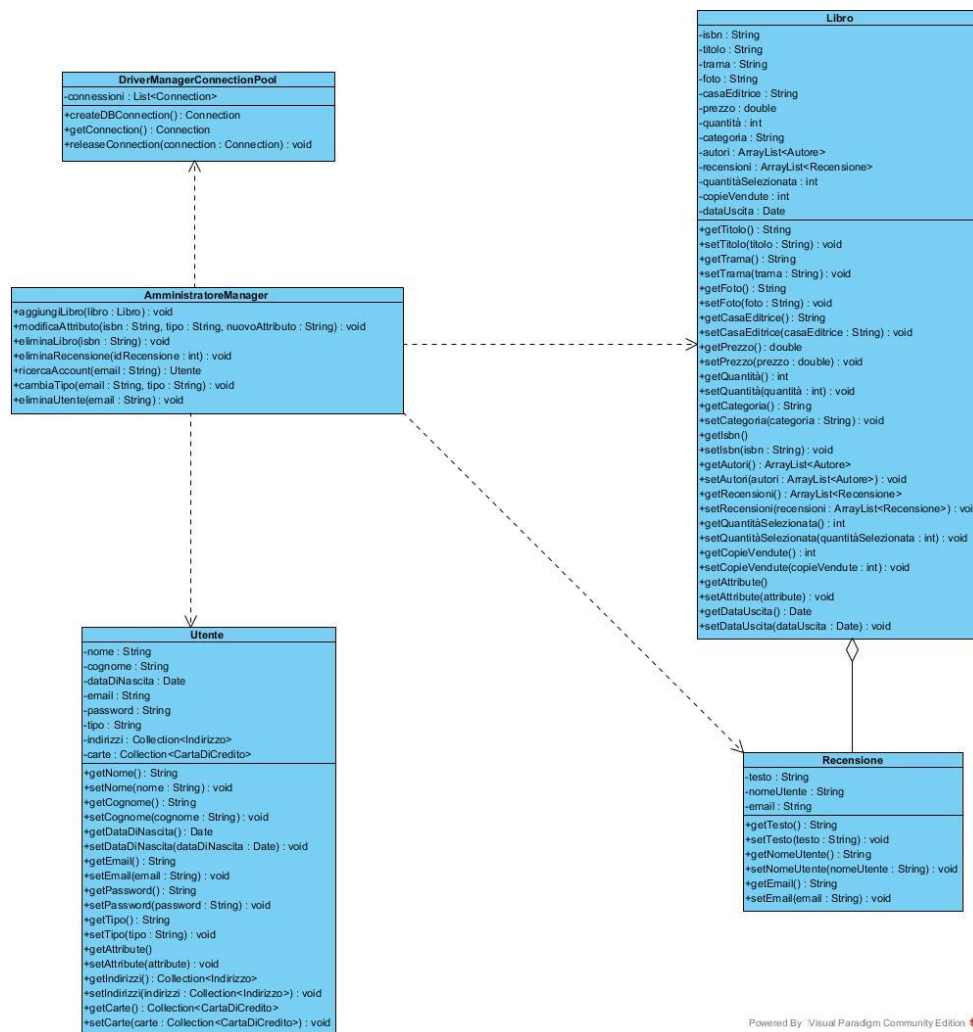


## 3.2 ACCOUNTMANAGER



Nome Classe	AccountManager
Pre-condizione	<b>context</b> AccountManager::login(email: String, password: String) : Utente <b>pre</b> email!=null && password != null; <b>context</b> AccountManager::registrazione(Utente utente) : boolean <b>pre</b> utente!=null; <b>context</b> AccountManager::changePassword(email: String, vecchiaPassword: String, nuovaPassword: String) : boolean <b>pre</b> email!= null && vecchiaPassword!= null && nuovaPassword!=null; <b>context</b> AccountManager::ricercaCarte(email : String) : Collection<CarteDiCredito> <b>pre</b> email!=null ; <b>context</b> AccountManager::ricercaIndirizzi(email : String) : Collection<Indirizzo> <b>pre</b> email!=null ; <b>context</b> AccountManager::aggiungiCarta(email : String, carta : CartaDiCredito) : void <b>pre</b> email!=null && utente!= null ; <b>context</b> AccountManager::aggiungiIndirizzo(email : String, indirizzo : Indirizzo) : void <b>pre</b> email!=null && indirizzo != null;
Post-condizione	<b>context</b> AccountManager::changePassword(email: String, vecchiaPassword: String, nuovaPassword: String) : boolean <b>post</b> utente.password=nuovaPassword <b>context</b> AccountManager::aggiungiCarta(email : String, carta : CartaDiCredito) : void <b>post</b> getCarte.size()= @pre.getCarte.size()+1 <b>context</b> AccountManager::aggiungiIndirizzo(email : String, carta : CartaDiCredito) : void <b>post</b> getIndirizzi.size()= @pre.getIndirizzi.size()+1
Invarianti	

### 3.3 AMMINISTRATOREMANAGER

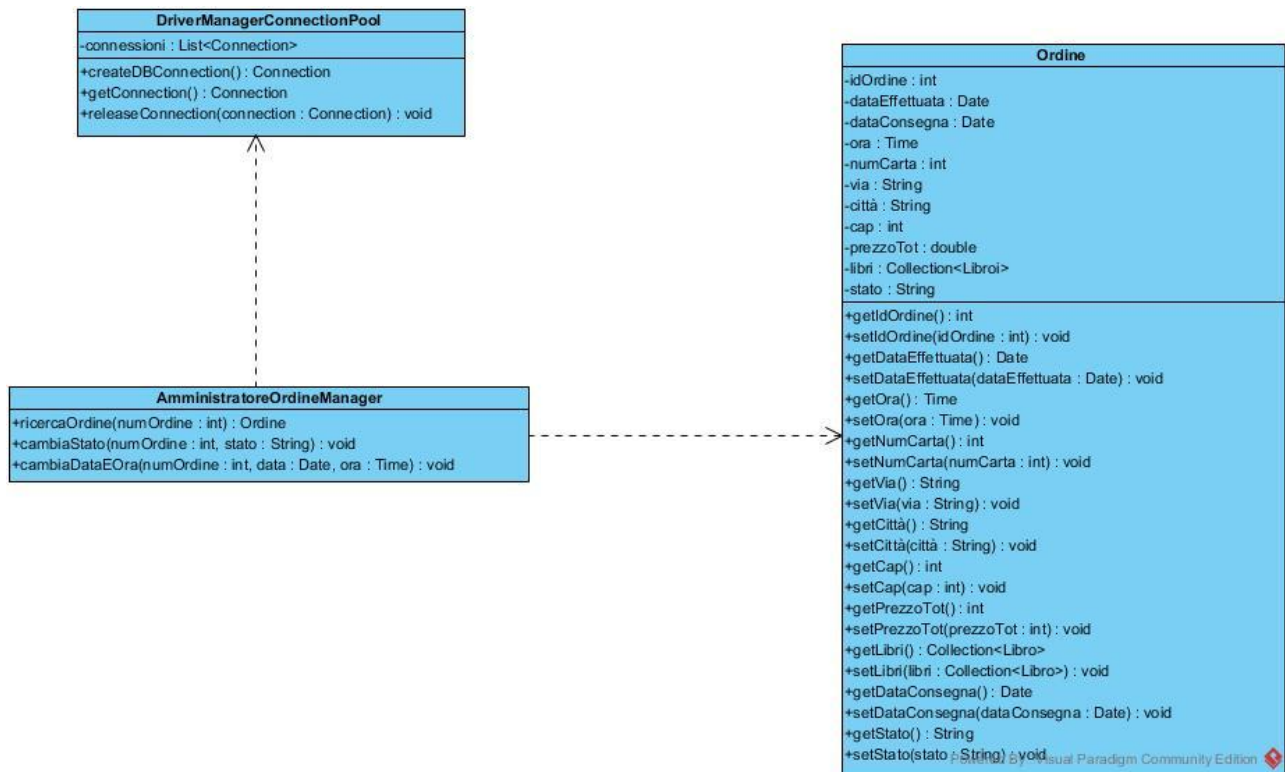


Nome Classe	AmministratoreManager
Pre-condizione	<p><b>context</b> AmministratoreManager::aggiungiLibro(libro : Libro) : void <b>pre:</b> libro!=null;</p> <p><b>context</b> AmministratoreManager::modificaTitolo(isbn : String, nuovoAttributo : String) : void <b>pre:</b> isbn!=null &amp;&amp; nuovoAttributo!= null;</p> <p><b>context</b> AmministratoreManager::modificaTrama(isbn : String, nuovoAttributo : String) : void <b>pre:</b> isbn!=null &amp;&amp; nuovoAttributo!= null;</p> <p><b>context</b> AmministratoreManager::modificaFoto(isbn : String, nuovoAttributo : String) : void <b>pre:</b> isbn!=null &amp;&amp; nuovoAttributo!= null;</p> <p><b>context</b> AmministratoreManager::modificaCasaEditrice(isbn : String, nuovoAttributo : String) : void <b>pre:</b> isbn!=null &amp;&amp; nuovoAttributo!= null;</p> <p><b>context</b> AmministratoreManager::modificaPrezzo(isbn : String, nuovoAttributo : Double) : void <b>pre:</b> isbn!=null &amp;&amp; nuovoAttributo!= null &amp;&amp; nuovoAttributo&gt;0;</p> <p><b>context</b> AmministratoreManager::modificaQuantità(isbn : String, nuovoAttributo : Int) : void <b>pre:</b> isbn!=null &amp;&amp; nuovoAttributo!= null &amp;&amp; nuovoAttributo&gt;0;</p> <p><b>context</b> AmministratoreManager::modificaCategoria(isbn : String, nuovoAttributo : String) : void <b>pre:</b> isbn!=null &amp;&amp; nuovoAttributo!= null;</p> <p><b>context</b> AmministratoreManager::modificaQuantitàSelezionata(isbn : String, nuovoAttributo : Int) : void <b>pre:</b> isbn!=null &amp;&amp; nuovoAttributo!= null &amp;&amp; nuovoAttributo&gt;0;</p> <p><b>context</b> AmministratoreManager::modificaCopieVendute(isbn : String, nuovoAttributo : Int) : void <b>pre:</b> isbn!=null &amp;&amp; nuovoAttributo!= null &amp;&amp; nuovoAttributo&gt;0;</p> <p><b>context</b> AmministratoreManager::modificaDataUscita(isbn : String, nuovoAttributo : Date) : void <b>pre:</b> isbn!=null &amp;&amp; nuovoAttributo!= null ;</p> <p><b>context</b> AmministratoreManager::eliminaLibro(isbn : String) : void <b>pre</b> isbn!=null;</p> <p><b>context</b> AmministratoreManager::eliminaRecensione(idRecensione : int) : void <b>pre</b> idRecensione!=null;</p> <p><b>context</b> AmministratoreManager::ricercaAccount(email : String) : Utente <b>pre</b> email!=null;</p>

	<b>context</b> AmministratoreManager::cambiaTipo(email : String, tipo : String) : void <b>pre</b> isbn!=null && tipo!=null; <b>context</b> AmministratoreManager::eliminaUtente(email : String) : void <b>pre</b> email!=null;
Post-condizione	<b>context</b> AmministratoreManager::aggiungiLibro(libro : Libro) : void <b>post</b> : libri.size() <u>=@pre.libri.size+1</u> <b>context</b> AmministratoreManager::modificaTitolo(isbn : String, nuovoAttributo : String) : void <b>post</b> : libro.titolo=nuovoAttributo; <b>context</b> AmministratoreManager::modificaTrama(isbn : String, nuovoAttributo : String) : void <b>post</b> : libro.trama=nuovoAttributo; <b>context</b> AmministratoreManager::modificaFoto(isbn : String, nuovoAttributo : String) : void <b>post</b> : libro.foto=nuovoAttributo; <b>context</b> AmministratoreManager::modificaCasaEditrice(isbn : String, nuovoAttributo : String) : void <b>post</b> : libro.casaEditrice=nuovoAttributo; <b>context</b> AmministratoreManager::modificaPrezzo(isbn : String, nuovoAttributo : Double) : void <b>post</b> : libro.prezzo=nuovoAttributo; <b>context</b> AmministratoreManager::modificaQuantità(isbn : String, nuovoAttributo : Int) : void <b>post</b> : libro.quantità=nuovoAttributo; <b>context</b> AmministratoreManager::modificaCategoria(isbn : String, nuovoAttributo : String) : void <b>post</b> : libro.categoria=nuovoAttributo; <b>context</b> AmministratoreManager::modificaQuantitàSelezionata(isbn : String, nuovoAttributo : Int) : void <b>post</b> : libro.quantitàSelezionata=nuovoAttributo; <b>context</b> AmministratoreManager::modificaCopieVendute(isbn : String, nuovoAttributo : Int) : void <b>post</b> : libro.copieVendute=nuovoAttributo; <b>context</b> AmministratoreManager::modificaDataUscita(isbn : String, nuovoAttributo : Date) : void <b>post</b> : libro.dataUscita=nuovoAttributo; <b>context</b> AmministratoreManager::eliminaLibro(isbn : String) : void <b>pre</b> isbn!=null; <b>post</b> : libri.size() <u>=@pre.libri.size+1</u> ; <b>context</b> AmministratoreManager::eliminaRecensione(idRecensione : int) : void <b>post</b> : libro.getRecensioni().size=@pre.libro.getRecensioni().size()-1; <b>context</b> AmministratoreManager::cambiaTipo(email : String, tipo : String) : void <b>post</b> : utente.getTipo=tipo;
Invarianti	

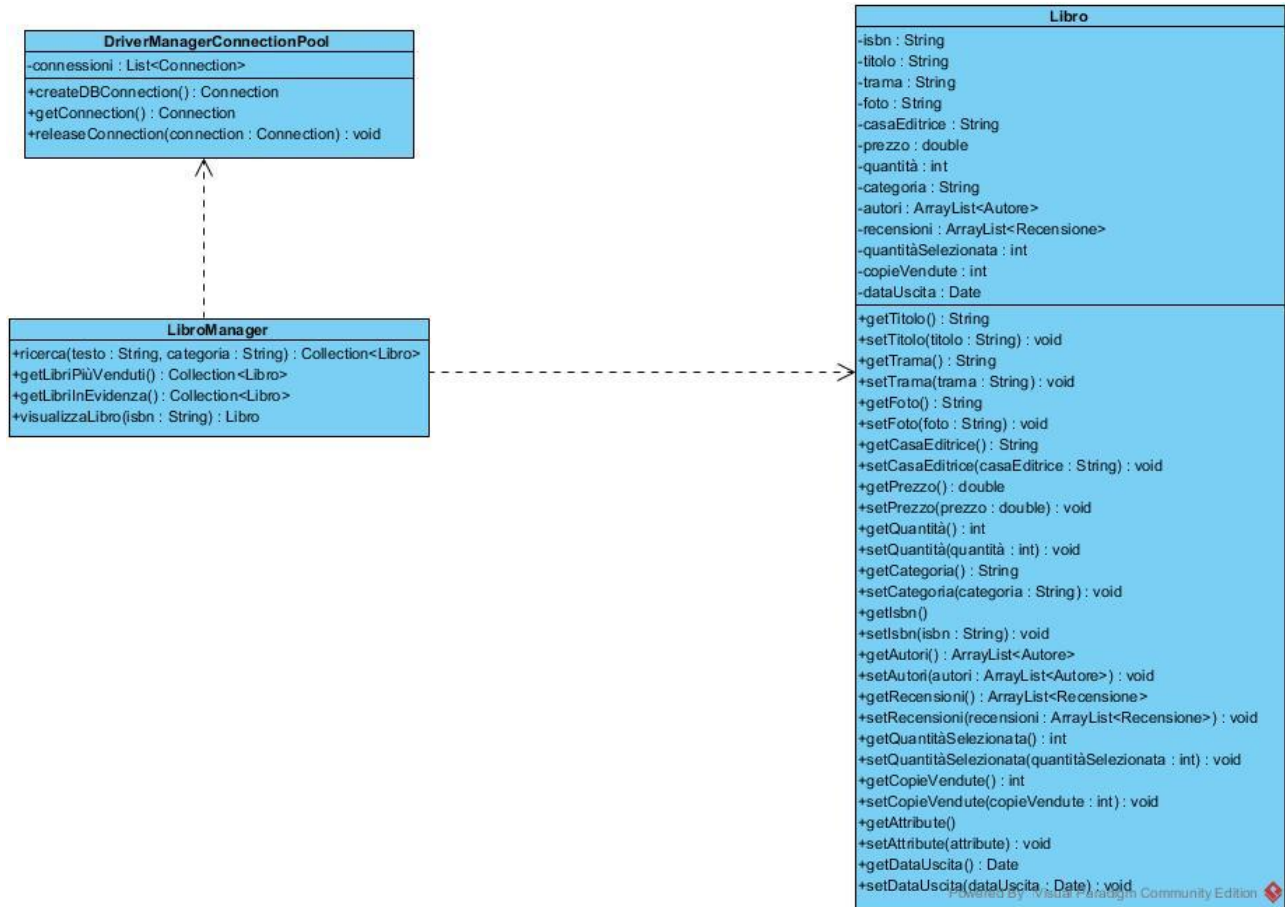


### 3.4 AMMINISTRATOREORDINIMANAGER



Nome Classe	AmministratoreOrdineManager
Pre-condizione	<b>context</b> AmministratoreOrdineManager::ricercaOrdine( numOrdine : int) : Ordine <b>pre</b> numOrdine!=null; <b>context</b> AmministratoreOrdineManager::cambiaStato(numOrdine : int, stato : String) : void <b>pre</b> numOrdine!=null && stato!=null; <b>context</b> AmministratoreOrdineManager::cambiaDataEOra(numOrdine : int, data : Date, ora : Time) : void <b>pre</b> numOrdine!=null && data!=null && ora!=null
Post-condizione	
Invarianti	

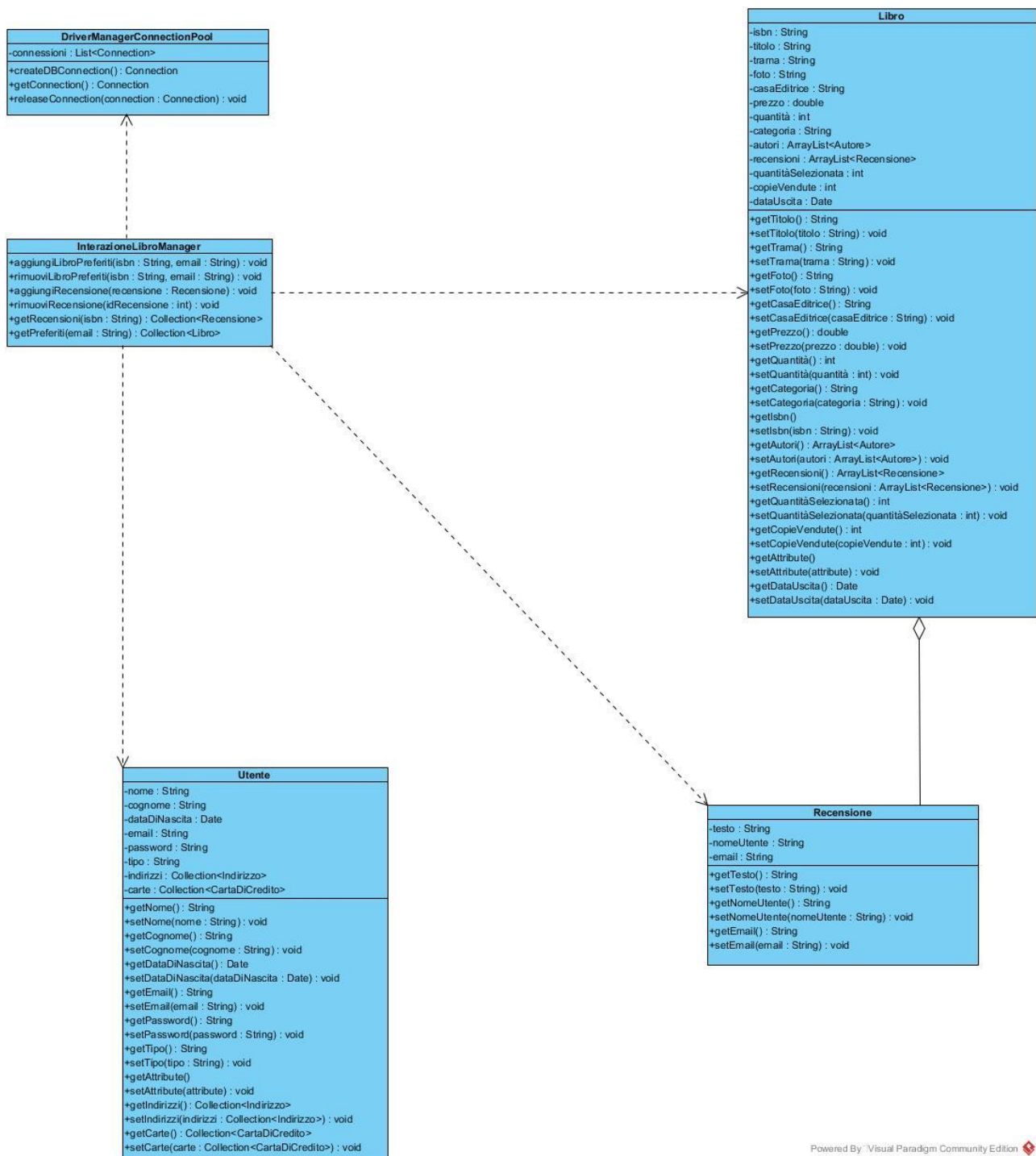
### 3.5 LIBROMANAGER



Nome Classe	LibroManager
Pre-condizione	<b>context</b> LibroManager::ricerca(testo : String, categoria : String) : Collection<Libro> <b>pre</b> : testo!=null && categoria!= null; <b>context</b> LibroManager::visualizzaLibro(isbn : String): Libro <b>pre</b> : isbn!=null
Post-condizione	
Invarianti	



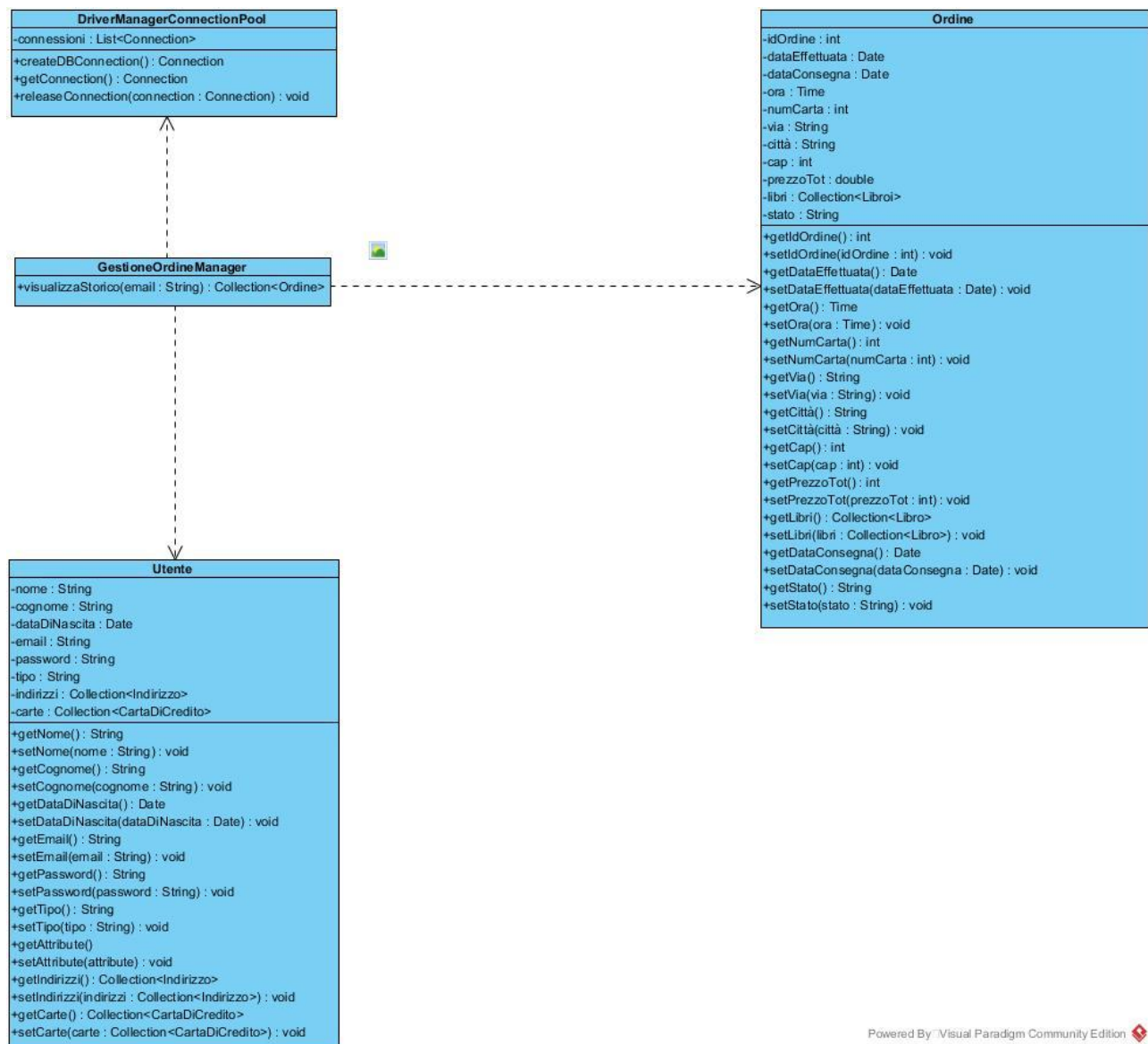
### 3.6 INTERAZIONE LIBROMANAGER



Nome Classe	InterazioneLibroManager
Pre-condizione	<b>context</b> InterazioneLibroManager::aggiungiLibroPreferiti(isbn : String, email: String): void <b>pre:</b> isbn!=null && email!=null; <b>context</b> InterazioneLibroManager::rimuoviLibroPreferiti(isbn : String, email: String) : void <b>pre:</b> isbn!=null && email!=null; <b>context</b> InterazioneLibroManager::aggiungiRecensione(recensione : Recensione): void <b>pre:</b> recensione!=null; <b>context</b> InterazioneLibroManager::rimuoviRecensione(idRecensione : int) : void <b>pre:</b> idRecensione!=null;

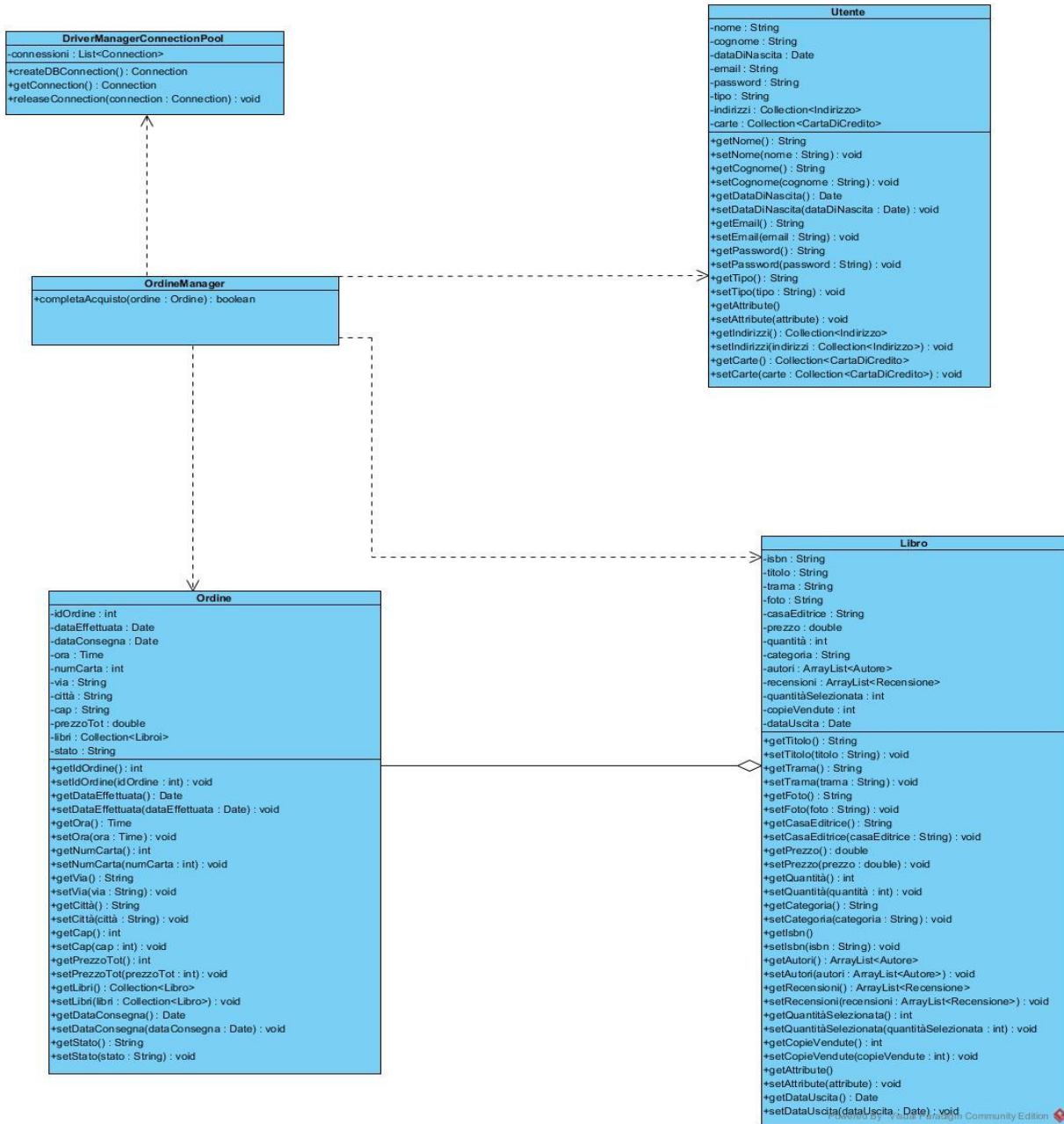
	<b>context</b> InterazioneLibroManager::getRecensioni(isbn : String) Collection<Libro> <b>pre:</b> isbn!=null; <b>context</b> InterazioneLibroManager::getPreferiti( email: String) Collection<Libro> <b>pre:</b> email!=null;
Post-condizione	
Invarianti	

### 3.7 GESTIONEORDINEMANAGER



Nome Classe	GestioneOrdineManager
Pre-condizione	<b>context</b> GestioneOrdineManager::visualizzaStorico(email : String): Collection<Libro> <b>pre:</b> email!=null
Post-condizione	
Invarianti	

### 3.8 ORDINEMANAGER



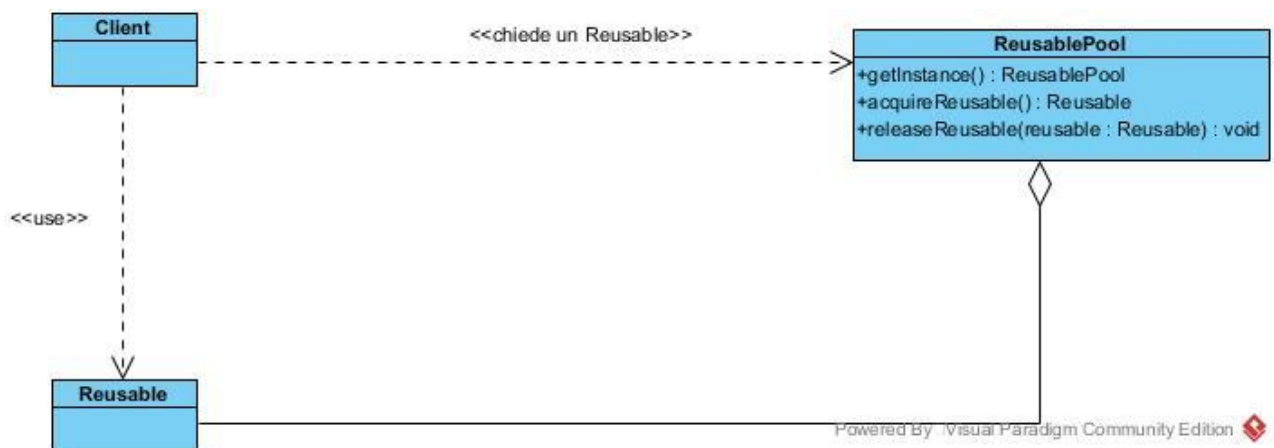
Nome Classe	OrdineManager
Pre-condizione	<b>context</b> OrdineManager::completaAcquisto(ordine : Ordine): boolean <b>pre:</b> ordine!=null; <b>context</b> OrdineManager::aggiungiAlCarrello(carrello : Carrello, isbn : String): Carrello <b>pre:</b> carrello!=null && isbn!=null; <b>context</b> OrdineManager::aumentaQuantità(carrello : Carrello, isbn : String): Carrello <b>pre:</b> carrello!=null && isbn!=null; <b>context</b> OrdineManager::diminuisciQuantità(carrello : Carrello, isbn : String): Carrello <b>pre:</b> carrello!=null && isbn!=null; <b>context</b> OrdineManager::rimuoviDalCarrello(carrello : Carrello, isbn : String): Carrello <b>pre:</b> carrello!=null && isbn!=null;
Post-condizione	<b>context</b> OrdineManager::rimuoviDalCarrello(carrello : Carrello, isbn : String): Carrello <b>post:</b> carrello.size()==@pre.carrello.size()-1;
Invarianti	

## 4 DESIGN PATTERN

### 4.1 OBJECT POOL PATTERN

L'Object pool pattern è un design pattern creazionale che usa un insieme di oggetti inizializzati pronti per l'uso mantenuti in una "pool", che si occupa di allocarli e de-allocherà su richiesta. Il client della pool invierà richiesta a un oggetto nella pool ed eseguirà operazioni sull'oggetto ritornato. Quando il client ha finito, ritorna l'oggetto alla pool che lo de-allocherà.

*Object pool pattern*



Utilizzo: L'Object Pool Pattern sarà utilizzato per gestire le connessioni con il database. Più precisamente, un oggetto Model richiederà connessioni al DriverManagerPool che ritornerà un oggetto Connection, l'oggetto Model effettuerà operazioni con l'oggetto connection e successivamente richiederà al DriverManagerPool la de-allocazione.

