Università degli studi di Salerno

Dipartimento di Informatica

***Corso di Laurea in Informatica***



***INGEGNERIA DEL SOFTWARE***

***Problem Statement***

***“VORWERK”***

*.*

**Docente:**

Andrea De Lucia

**Studenti:**

##### Nome Matricola

Vincenzo Marrazzo 0512105832

*Anno Accademico: 2020/21*

INDICE

Sommario

[**1.** **Introduzione** 2](#_Toc63764536)

[1.1 Object Design trade-offs 3](#_Toc63764537)

[1.2 Linee Guida per la Documentazione delle Interfacce 3](#_Toc63764538)

[1.3 Definizioni, acronimi e abbreviazioni 5](#_Toc63764539)

[1.4 Design Pattern 5](#_Toc63764540)

[1.4.1 Data Access Object 5](#_Toc63764541)

[**2.** **Packages** 5](#_Toc63764542)

[2.1 Gestione Robot 6](#_Toc63764543)

[2.2 Gestione Account 7](#_Toc63764544)

[2.3 Gestione Premi 8](#_Toc63764545)

[2.4 Gestione Acquisto Robot 9](#_Toc63764546)

[2.5 Gestione Ricevimento Premi 10](#_Toc63764547)

[**3.** **Class Interfaces** 11](#_Toc63764548)

[3.1 Gestione Robot 12](#_Toc63764549)

[3.2 Gestione Account 13](#_Toc63764550)

[3.3 Gestione Premi 16](#_Toc63764551)

[3.4 Gestione Acquisto Robot 18](#_Toc63764552)

[3.5 Gestione Ricevimento Premi 20](#_Toc63764553)

# **Introduzione**

L’object design è il processo che si occupa di aggiungere dettagli all’analisi dei requisiti e di prendere decisioni di implementazione. I progettisti devono scegliere i diversi modi per implementare il modello di analisi, con l’obbiettivo di minimizzare il tempo di esecuzione, la memoria e altre misure di costo. In particolar modo, in tale documento si definiscono 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.

## Object Design trade-offs

**Comprensibilità vs Tempo:**  
Il codice del sistema deve essere comprensibile il più possibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare.

**Prestazione vs Costi:**Verranno utilizzati dei template open source esterni, in particolare Bootstrap, visto che per la creazione di questo progetto non viene istanziato nessun budget e allo stesso tempo vogliamo poter mantenere le giuste prestazioni.

**Interfaccia vs Usabilità:**L’interfaccia grafica si basa sull’utilizzo di form e pulsanti e sono stati realizzati in maniera molto semplice e chiara, con lo scopo di rendere semplice l’utilizzo del sistema da parte dell’utente finale.

**Sicurezza vs Efficienza:**La sicurezza, come descritto nei requisiti non funzionali del Requirements Analysis Document, rappresenta uno degli aspetti importanti del sistema. Tuttavia, dati i tempi di sviluppo molto limitati, mi limiterò ad implementare sistemi di sicurezza basati su e-mail e password degli utenti.

## Linee Guida per la Documentazione delle Interfacce

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

|  |  |  |
| --- | --- | --- |
| Tipi | Regole | Esempi |
| Packages | La prima lettera della prima parola del nome di una classe deve avere lettera minuscola, mentre le parole successive devono avere la prima lettera minuscola | pacchetto |
| Classes | I nomi delle classi devono iniziare con la lettera maiuscola, e anche le parole successive all’interno del nome devono iniziare con la lettera maiuscola(“GobbaDiCammello”). I nomi delle classi devono essere evocativi, in modo da fornire informazioni sullo scopo di quest’ultime | ManagerRobot |
| Methods | I nomi dei metodi devono iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola. Di solito il nome del metodo è costituito da un verbo che identifica un’azione, seguito dal nome di un oggetto.  Per quanto riguarda i nomi dei metodi che vengono utilizzati o per l’accesso o per la modifica, dovranno essere rispettivamente del tipo getNomeVariabile() e setNomeVariabile() | setNomeVariabile(); getNomeVariabile(); doSomething(); |
| Variables | I nomi delle variabili devono iniziare con la lettera minuscola, e le parole successive con la lettera iniziale maiuscola. In determinati casi è possibile utilizzare il carattere underscore “\_”, ad esempio quando si fa uso di variabili costanti oppure quando si fa uso di proprietà statiche | variabile; v; idRobot; QUANTITA\_DISPONIBILE; |

## Definizioni, acronimi e abbreviazioni

* E-mail = e-mail con cui accede al Sistema l’utente
* Password = chiave per accedere al sistema
* mySql = programma per la gestione del database

## Design Pattern

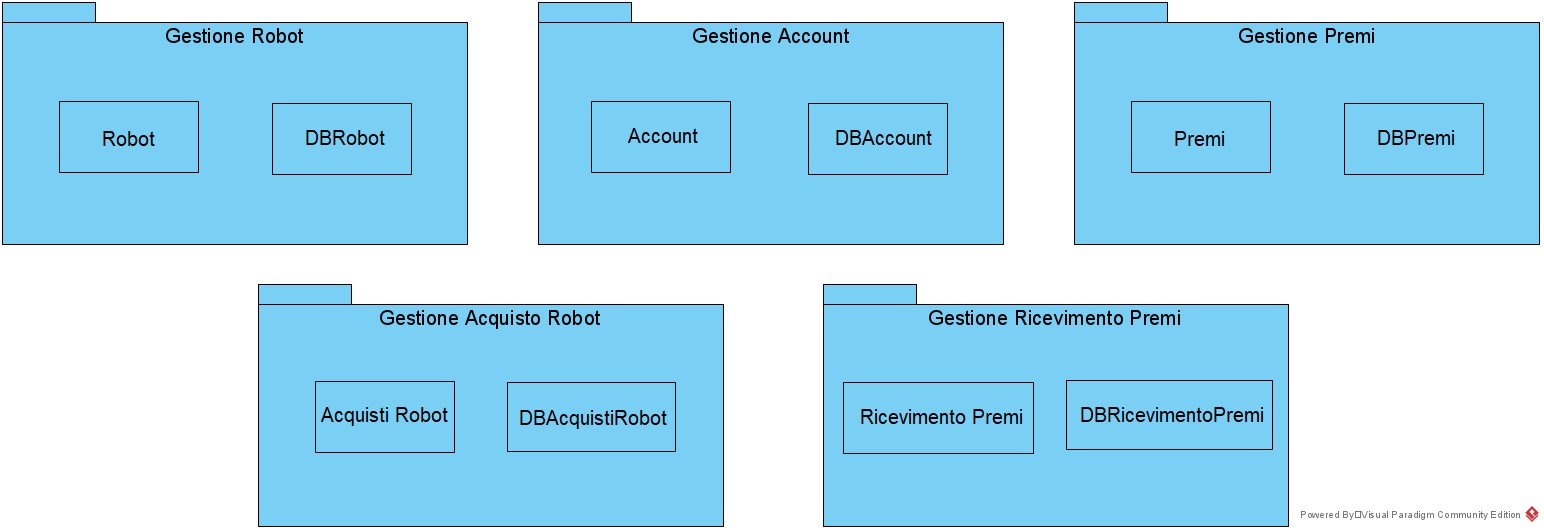
### Data Access Object

In informatica, nell’ambito della programmazione Web, il DAO (Data Access Object) è un pattern architetturale. Si tratta fondamentalmente di una classe con relativi metodi che rappresenta un’entità tabellare di un RDBMS, usata principalmente in applicazioni web sia di tipo di Java EE sia di tipo EJB, per stratificare e isolare l’accesso ad una tabella tramite query(poste all’interno dei metodi della classe) ovvero al data layer da parte della business logic creando un maggiore livello di astrazione ed una più manutenibilità. I metodi del DAO con le rispettive query(cioè le query che si trovano all’interno dei metodi) verranno così richiamati dalle classi della business logic.

# **Packages**

Le funzionalità elencate saranno implementate nel mio sistema:

* Gestione Robot
* Gestione Account
* Gestione Premi
* Gestione Acquisto Robot
* Gestione Ricevimento Premi

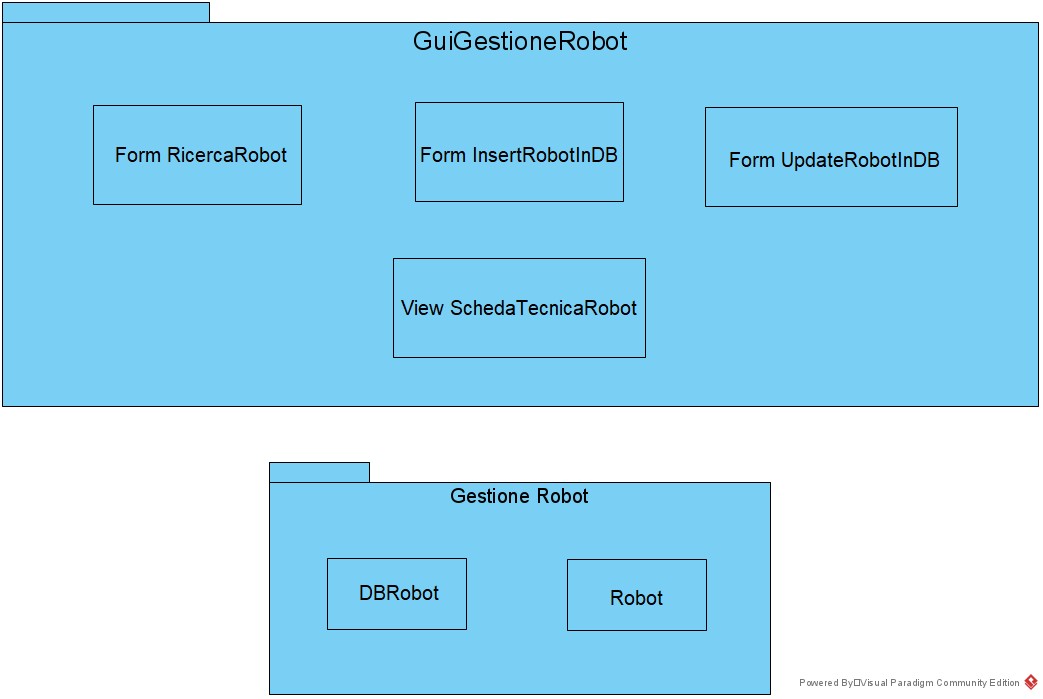
Di seguito, viene riportata una prima visione grafica, successivamente raffinata, delle funzionalità implementate  
 

Nel successivo paragrafo vengono elencate graficamente le classi che compongono ciascun package.  
Una loro descrizione approfondita (metodi e campi) è fornita al punto 3.

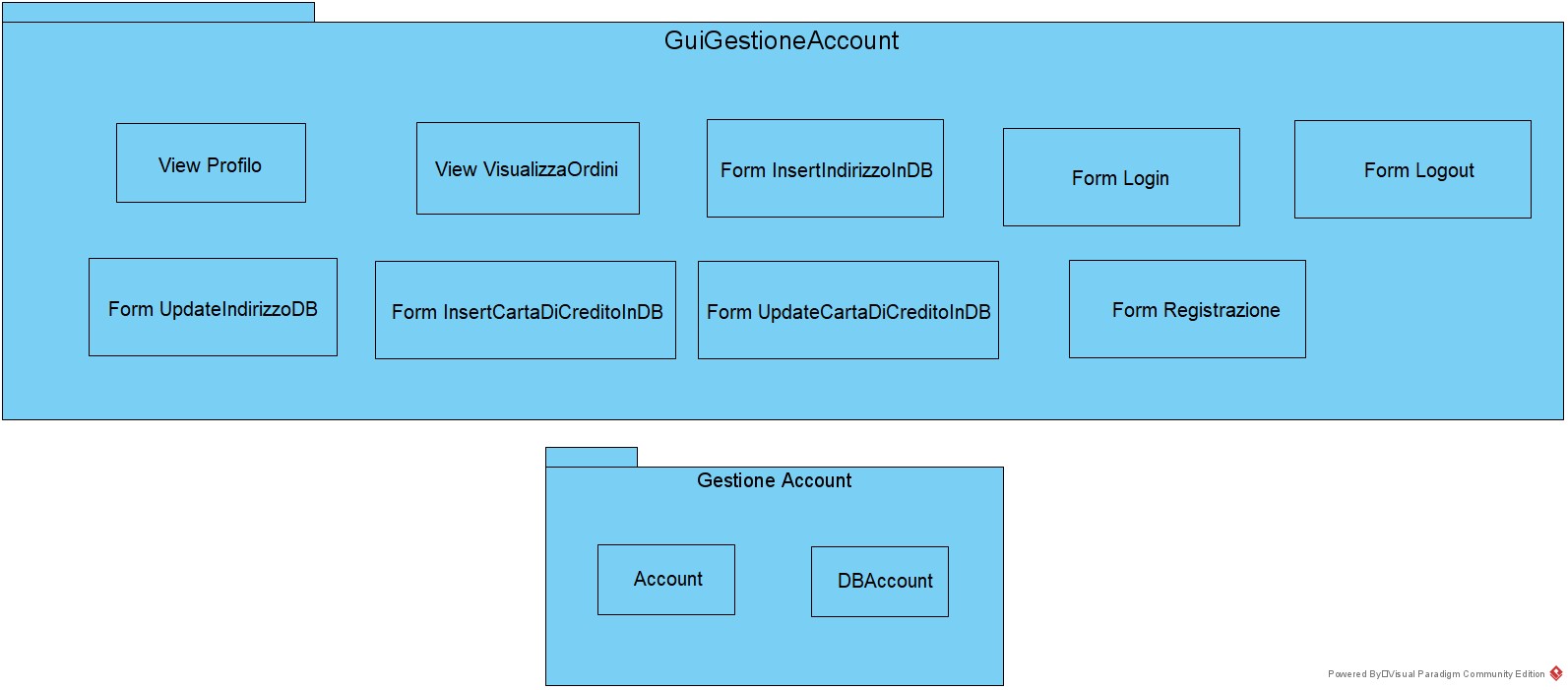
Per ciascun gruppo di funzionalità, sono presenti, quindi, due pacchetti, ognuno composto da diverse classi:

* GestioneXxx, che si occupa, appunto, della logica applicativa e della memorizzazione dei dati.

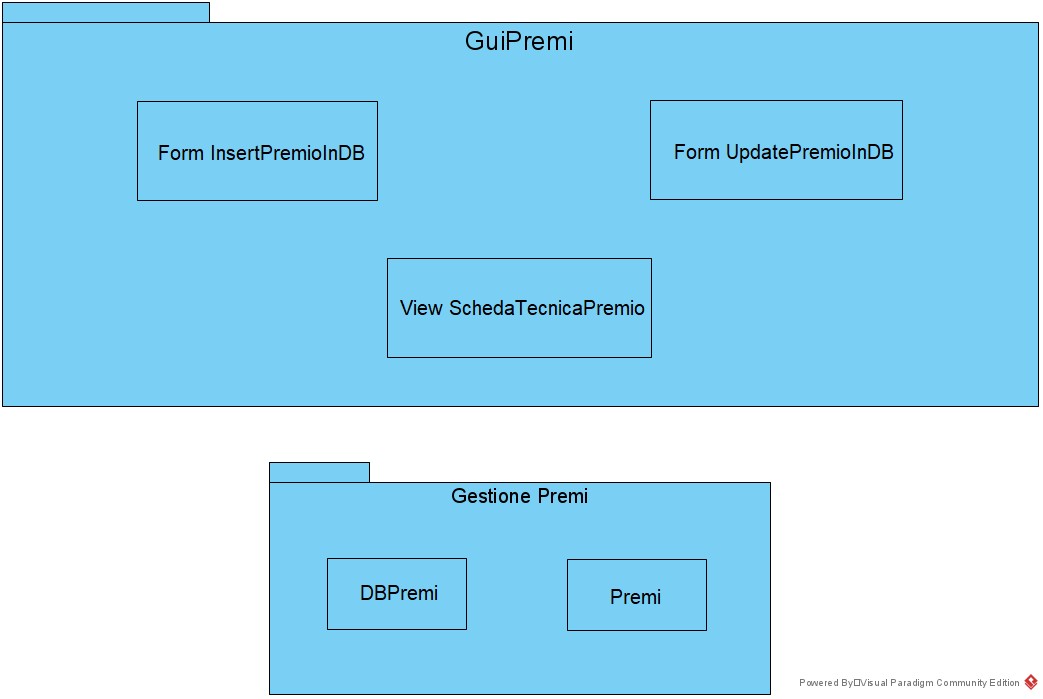
## Gestione Robot



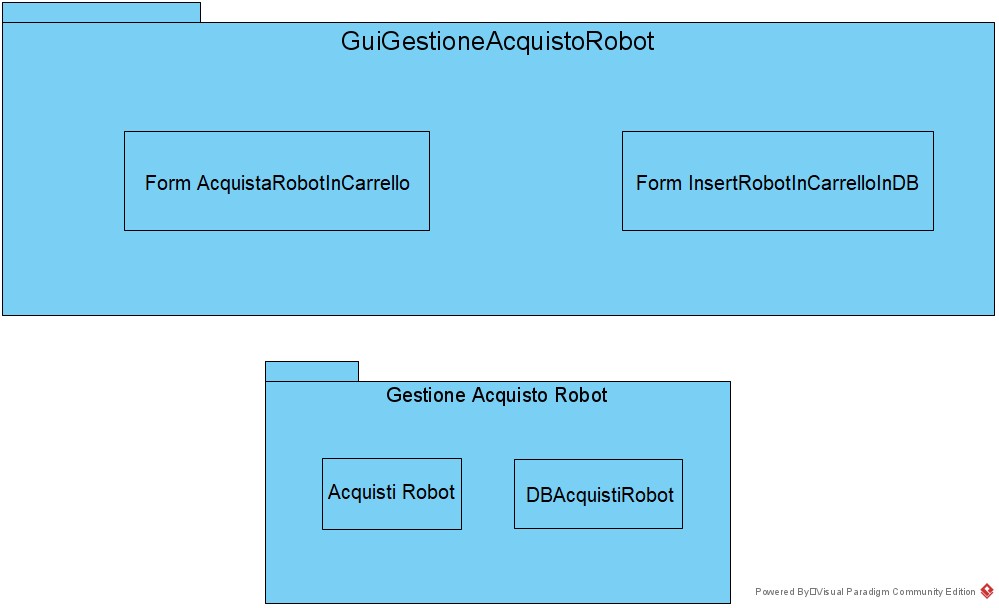
## Gestione Account



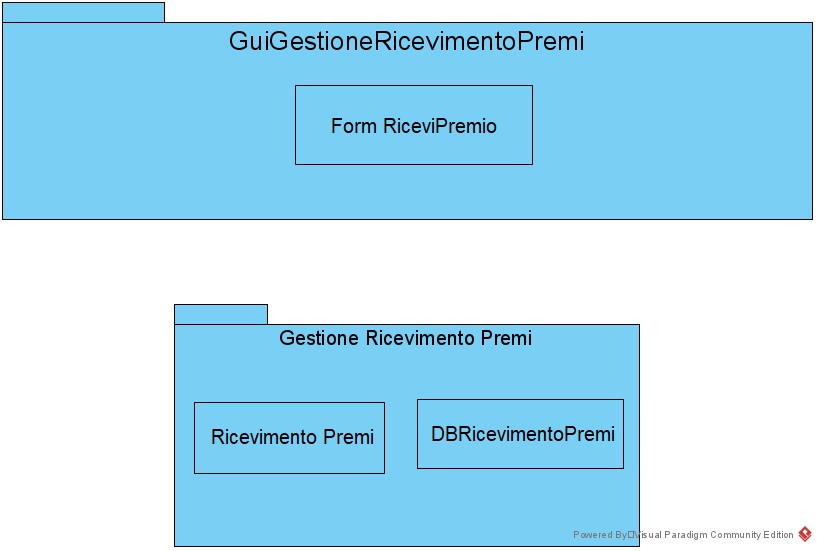
## Gestione Premi



## Gestione Acquisto Robot

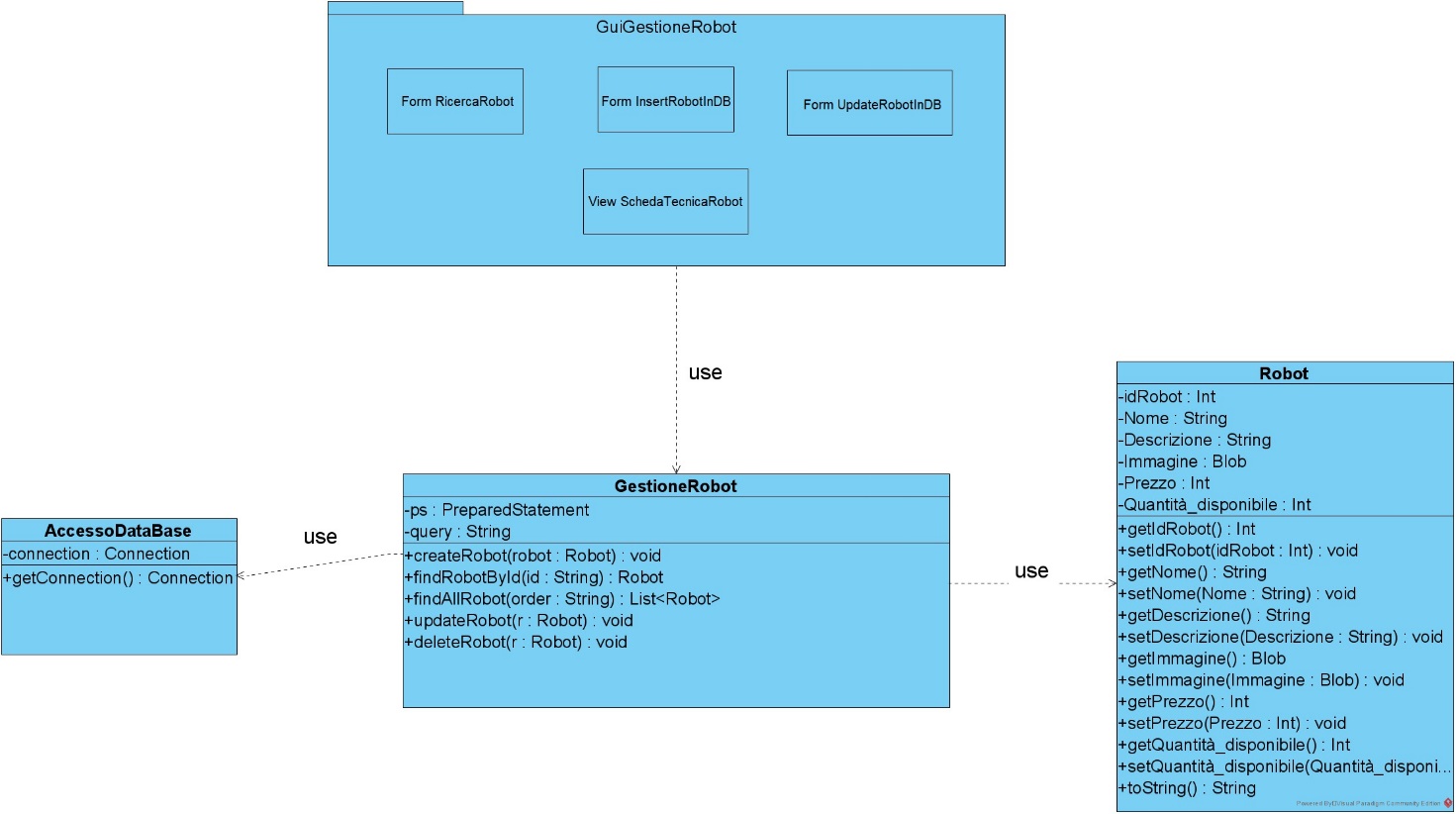


## Gestione Ricevimento Premi



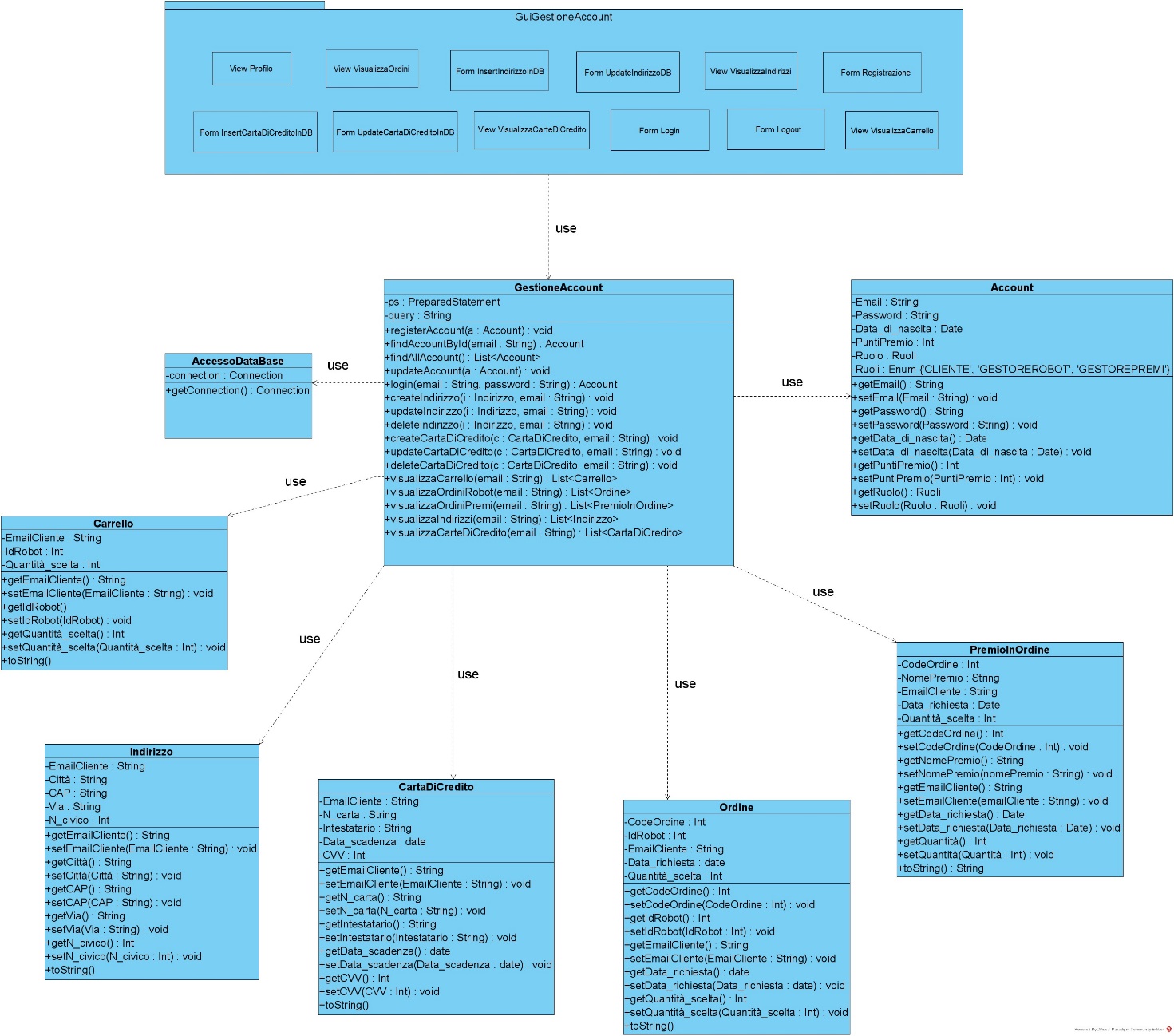
# **Class Interfaces**

## Gestione Robot



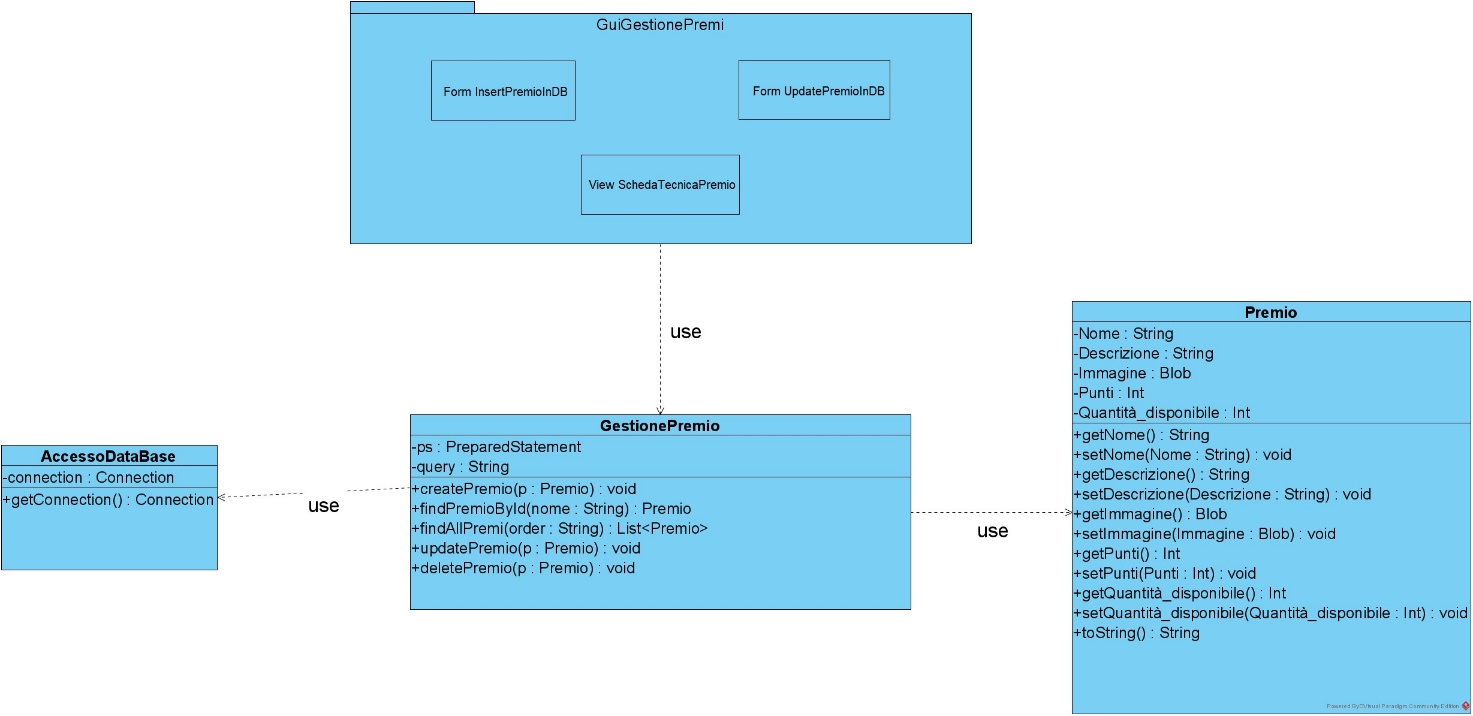
|  |  |
| --- | --- |
| Nome | Gestione Robot |
| Descrizione | Gestione Robot permette la gestione dei robot del database(aggiunta/modifica/rimozione), ricerca e visualizzazione scheda tecnica. |
| Pre-condizione | Context GestioneRobot::createRobot(robot : Robot) : void  Pre:  Tutte le variabili d’istanza della classe Robot non devono essere null tranne per l’immagine  Context GestioneRobot::findRobotById(id : String) : Robot  Pre:  id<>null  Context GestioneRobot::findAllRobot(order : String) : List<Robot>  Pre:  order<>null and (order.equals(“idRobot”) OR order.equals(“Nome”) OR order.equals(“Descrizione”))  Context GestioneRobot::updateRobot(r : Robot) : void  Pre:  Tutte le variabili d’istanza della classe Robot non devono essere null tranne per l’immagine  Context GestioneRobot::deleteRobot(r : Robot) : void  Pre:  Tutte le variabili d’istanza della classe Robot non devono essere null tranne per l’immagine |
| Post-condizione | Context GestioneRobot::findRobotById(id : int) : Robot  Post:  Ritorna un Robot che ha Robot.idRobot uguale all’id del parametro esplicito  Context GestioneRobot::findAllRobot(order : String) : List<Robot>  Post:  Ritorna la lista ordinata di tutti i Robot |
| Invarianti |  |

## Gestione Account



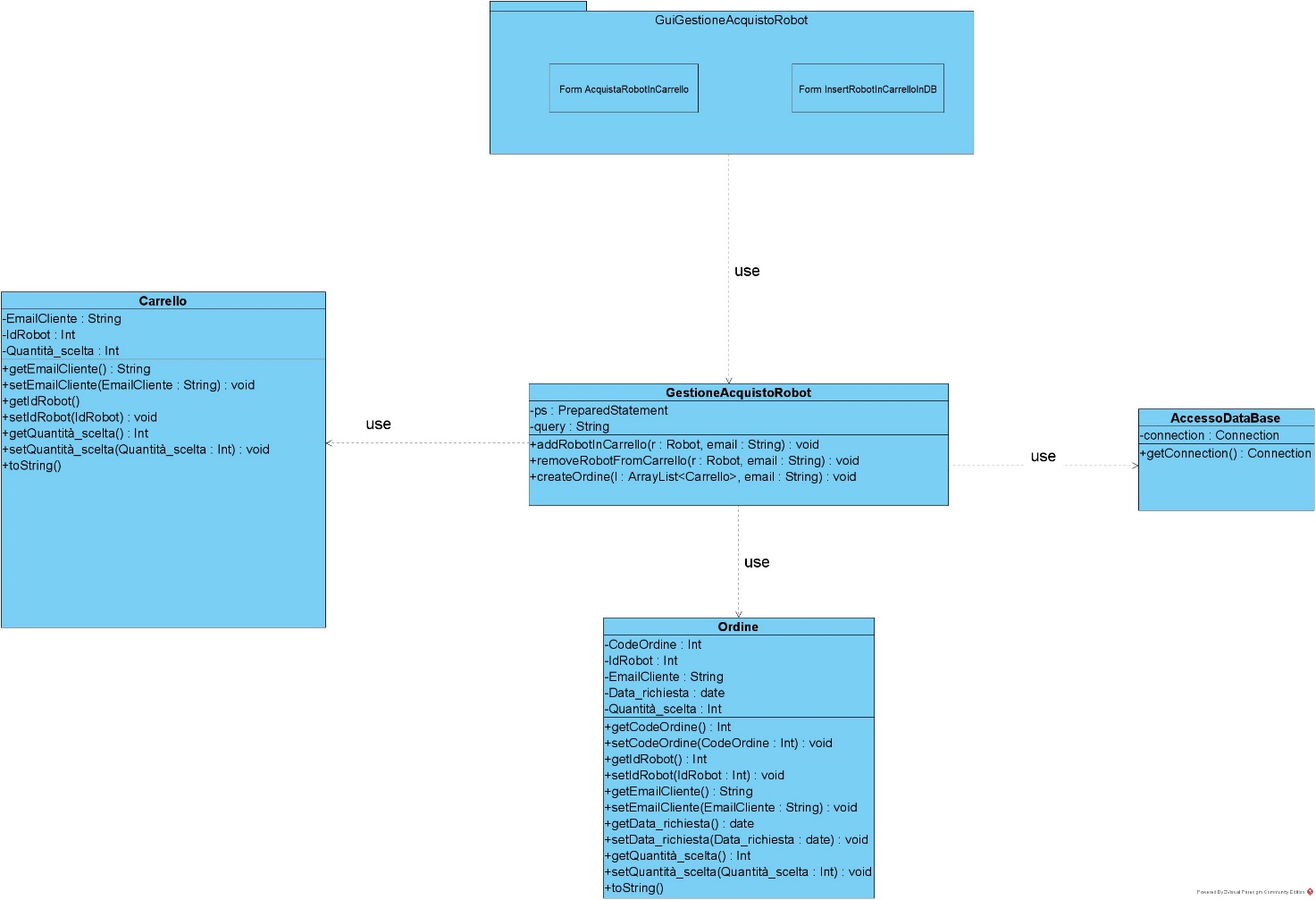
|  |  |
| --- | --- |
| Nome | Gestione Account |
| Descrizione | Gestione Account permette la gestione degli account del database(modifica), gestione e visualizzazione del profilo e visualizzazione del carrello. |
| Pre-condizione | Context GestioneAccount::registerAccount(a : Account) : void  Pre:  Tutte le variabili d’istanza della classe Account non devono essere null  Context GestioneAccount::findAccountById(email : String) : Account  Pre:  email<>null  Context GestioneAccount::updateAccount(a : Account) : void  Pre:  Tutte le variabili d’istanza della classe Account non devono essere null  && a.email deve esistere  Context GestioneAccount::login(email : String, password : String) : Account  Pre:  email<>null and password<>null  Context GestioneAccount::createIndirizzo(i : Indirizzo, email : String) : void  Pre:  indirizzo<>null and email<>null  Context GestioneAccount::updateIndirizzo(i : Indirizzo, email : String) : void  Pre:  Tutte le variabili d’istanza della classe Indirizzo non devono essere null  && email<>null  Context GestioneAccount::deleteIndirizzo(i : Indirizzo, email : String) : void  Pre:  Tutte le variabili d’istanza della classe Indirizzo non devono essere null  && email<>null  Context GestioneAccount::createCartaDiCredito(c : CartaDiCredito, email : String) : void  Pre:  Tutte le variabili d’istanza della classe CartaDiCredito non devono essere null  && email<>null  Context GestioneAccount::updateCartaDiCredito(c : CartaDiCredito, email : String) : void  Pre:  Tutte le variabili d’istanza della classe CartaDiCredito non devono essere null  && email<>null  Context GestioneAccount::deleteCartaDiCredito(c : CartaDiCredito, email : String):void  Pre:  Tutte le variabili d’istanza della classe CartaDiCredito non devono essere null  && email<>null  Context GestioneAccount::visualizzaCarrello(email : String): List<Robot>  Pre:  email<>null  Context GestioneAccount::visualizzaOrdiniRobot(email : String): List<Ordine>  Pre:  email<>null  Context GestioneAccount::visualizzaOrdiniPremi(email : String): List<PremioInOrdine>  Pre:  email<>null  Context GestioneAccount::visualizzaIndirizzi(email : String): List<Indirizzo>  Pre:  email<>null  Context GestioneAccount::visualizzaCarteDiCredito(email : String): List<CartaDiCredito>  Pre:  email<>null |
| Post-condizione | Context GestioneAccount::findAccountById(email : String) : Account  Post:  Ritorna un account dove account.email corrisponde con l’email inserita nel parametro esplicito  Context GestioneAccount::login(email : String, password : String) : Account  Post:  Ritorna un account  Context GestioneAccount::visualizzaCarrello(email : String): List<Robot>  Post:  Ritorna la lista di Robot che si trova nel carrello dell’email indicata come parametro esplicito  Context GestioneAccount::visualizzaOrdiniRobot(email : String): List<Ordine>  Post:  Ritorna la lista di Ordini di Robot dell’email indicata  Context GestioneAccount::visualizzaOrdiniPremi(email : String): List<PremioInOrdine>  Post:  Ritorna una lista di Ordini dei Premi dell’email indicata  Context GestioneAccount::visualizzaIndirizzi(email : String): List<Indirizzo>  Post:  Ritorna una lista di Indirizzi dell’email indicata  Context GestioneAccount::visualizzaCarteDiCredito(email : String): List<CartaDiCredito>  Post:  Ritorna una lista di Carte di Credito dell’email indicata |
| Invarianti |  |

## Gestione Premi



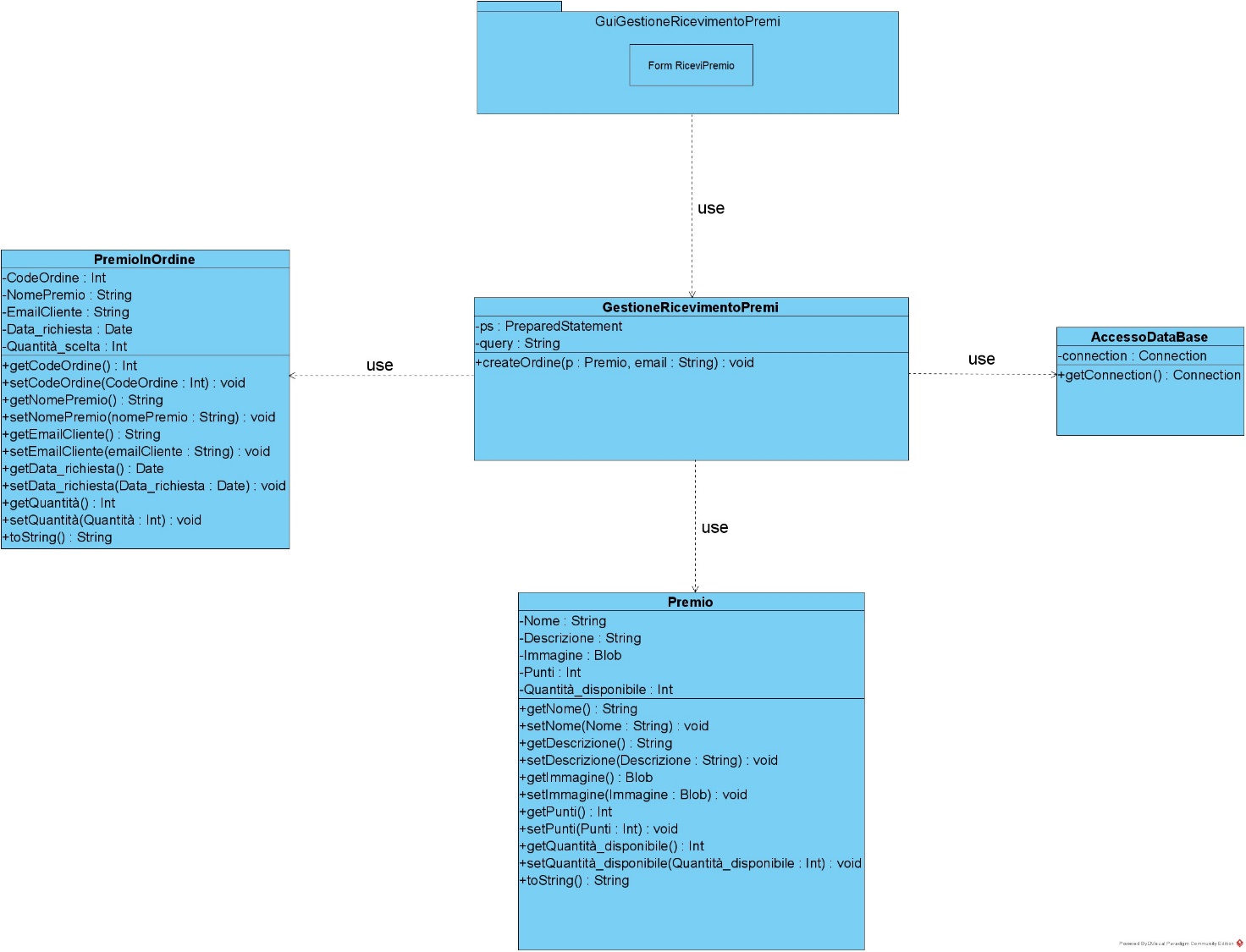
|  |  |
| --- | --- |
| Nome | Gestione Premi |
| Descrizione | Gestione Premi permette la gestione dei premi del database(aggiunta/modifica/rimozione) e visualizzazione scheda tecnica. |
| Pre-condizione | Context GestionePremi::createPremio(p : Premio) : void  Pre:  p<>null && Tutte le variabili d’istanza della classe Premio non devono essere null  Context GestionePremi::findPremioById(nome : String) : Premio  Pre:  nome<>null  Context GestionePremi::findAllPremi(order : String) : List<Premio>  Pre:  order<>null  && (order.equals(“Nome”), order.equals(“Descrizione”), order.equals(“Punti”), order.equals(“Quantità\_disponibile”),  Context GestionePremi::updatePremio(p : Premio) : void  Pre:  p <>null && Tutte le variabili d’istanza della classe Premio non devono essere null  Context GestionePremi::deletePremio(p : Premio) : void  Pre:  p <>null && Tutte le variabili d’istanza della classe Premio non devono essere null |
| Post-condizione | Context GestionePremi::findPremioById(nome : String) : Premio  Post:  Ritorna il Premio trovato  Context GestionePremi::findAllPremi(order : String) : List<Premio>  Post:  Ritorna una lista di Premi |
| Invarianti |  |

## Gestione Acquisto Robot



|  |  |
| --- | --- |
| Nome | Gestione Acquisto Robot |
| Descrizione | Gestione Acquisto Robot permette la gestione dell’acquisto di un robot, cioè aggiunta/rimozione di robot dal carrello e conferma dell’acquisto dei robot presenti nel carrello(cioè crea l’ordine per i robot che si intende comprare ovviamente dopo aver confermato l’acquisto). |
| Pre-condizione | Context GestioneAcquistoRobot::addRobotInCarrello(r : Robot, email : String) : void  Pre:  r <> null  && Tutte le variabili d’istanza della classe Robot non devono essere null and email<>null  Context GestioneAcquistoRobot::removeRobotFromCarrello(r : Robot, email : String) : void  Pre:  r <> null  && Tutte le variabili d’istanza della classe Robot non devono essere null  && email<>null  Context GestioneAcquistoRobot::createOrdine(l : List<Carrello>, email : String) : void  Pre:  l <> null  && email<>null |
| Post-condizione |  |
| Invarianti |  |

## Gestione Ricevimento Premi



|  |  |
| --- | --- |
| Nome | Gestione Ricevimento Premi |
| Descrizione | Gestione Ricevimento Premi permette la gestione del ricevimento di un premio, cioè vi è la conferma del ricevimento di un premio presente nel catalogo e quindi viene creato l’ordine con il suddetto premio(si controlla se il cliente ha i punti necessari per continuare con la compera del premio). |
| Pre-condizione | Context GestioneRicevimentoPremi::createOrdine(p : Premio, email : String) : void  Pre:  p<>null  && Tutte le variabili d’istanza della classe Premio non devono essere null  && email<>null |
| Post-condizione |  |
| Invarianti |  |