|  |
| --- |
| Università degli Studi di Salerno  Corso di Ingegneria del Software |

Kawaii-Comix

Object Design Document

Versione 1.1

Immagine che contiene fiore, rosa, Elementi grafici, testo

Descrizione generata automaticamente

Data: 10/01/2024

**Coordinatore del progetto**

**Partecipanti**

|  |  |
| --- | --- |
| Nome | Matricola |
| Simone D’Assisi | 0512113584 |
| Davide Del Franco Natale | 0512113233 |
| Giovanni Sicilia | 0512107458 |

|  |  |
| --- | --- |
| **Scritto da:** | Simone D’Assisi, Davide Del Franco Natale, Giovanni Sicilia |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autore** |
| 09/01/2024 | 1.0 | Prima stesura del documento | Simone D’Assisi |
| 10/01/2024 | 1.1 | Completamento prima stesura | Simone D’Assisi |

**Indice**

[1. Introduzione 4](#_Toc155720308)

[*1.1 Object design trade-offs* 4](#_Toc155720309)

[*1.1.1 Robustezza vs Tempo* 4](#_Toc155720310)

[*1.1.2 Attendibilità vs Tempo* 4](#_Toc155720311)

[*1.2 Linee guida* 4](#_Toc155720312)

[*1.3 Referenze* 4](#_Toc155720313)

[2. Directory 4](#_Toc155720314)

[*2.1* *src* 4](#_Toc155720315)

[2.2 *db* 5](#_Toc155720316)

[3. Pacchetti 5](#_Toc155720317)

[*3.1 view* 5](#_Toc155720318)

[3.1.1 view.site 5](#_Toc155720319)

[3.1.2 view.acquisto 6](#_Toc155720320)

[3.1.3 view.utente 6](#_Toc155720321)

[3.1.4 view.catalogo 7](#_Toc155720322)

[*3.2 acquistoManagement* 8](#_Toc155720323)

[*3.3 utenteManagement* 9](#_Toc155720324)

[*3.4 catalogoManagement* 10](#_Toc155720325)

[4. Interfacce di classe 11](#_Toc155720326)

# 1. Introduzione

## *1.1 Object design trade-offs*

### *1.1.1 Robustezza vs Tempo*

Il controllo dei dati in input è un aspetto importante del sistema, farlo però nella maniera più corretta e completa possibile richiederebbe un tempo di sviluppo maggiore rispetto a quello a nostra disposizione per il rilascio di una prima versione completamente funzionante. A discapito della robustezza, dunque, decidiamo di limitare i controlli sui dati in input, cosa che però sarà fatta nelle versioni successive.

### *1.1.2 Attendibilità vs Tempo*

L’attendibilità è un requisito fondamentale per il funzionamento del sistema, è infatti, ad esempio, importante gestire le transazioni in maniera corretta evitando che gli ordini possano essere effettuati senza avere la disponibilità dei prodotti richiesti. Chiaramente questo requisito necessita di maggior tempo di sviluppo per essere garantito, è infatti grazie al tempo guadagnato a discapito della robustezza che ci possiamo permettere di rispettarlo.

## *1.2 Linee guida*

Qui di seguito sono riportate alcune linee guide per la stesura del codice:

* Gli oggetto DAO devono essere nominati nomeentitàIDS.
* Gli errori devono essere gestiti tramite delle eccezioni e non dei valori di ritorno.

## *1.3 Referenze*

* **R.A.D.**
* **S.D.D.**

# 2. Directory

## *src*

* + 1. java

Questa directory deve contenere i pacchetti Java che compongono i codice del sistema

* + 1. test

Questa directory contiene i pacchetti java e i file necessari per il testing del sistema

* + 1. webapp
       1. META-INF

Questa directory deve contenere le meta-informazioni.

* + - 1. Script

Questa directory contiene i file JavaScript con numerose funzioni utilizzate dalle jsp.

* + - 1. Icons

Questa directory deve contenere le immagini utilizzate per icone o sfondi.

* + - 1. Images

Questa directory deve contenere le immagini utilizzate per le anteprime dei prodotti.

* + - 1. Styles

Questa directory deve contenere i fogli di stile.

* + - 1. Video

Questa directory deve contenere i video di presentazione.

## *db*

Questa directory deve contenere lo schema del database relazionale.

# 3. Pacchetti

In questa sezione analizzeremo la suddivisione in pacchetti di tutte le classi implementate, ecco una prima panoramica del sistema implementato:

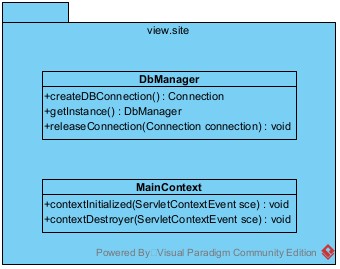
## 

## *3.1 view*

Questo pacchetto contiene tutti gli oggetti e le classi Java che compongono il layer di presentation del sistema.

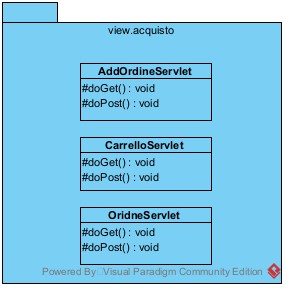
### 3.1.1 view.site

Questo sottopacchetto contiene le classi Java adibite alla connessione al Database.



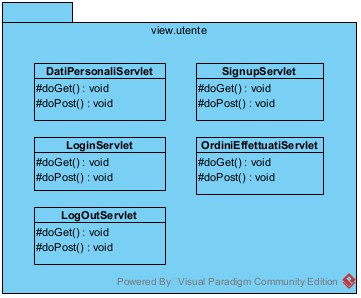
### 3.1.2 view.acquisto

Questo sottopacchetto contiene le Servlet adibite alle funzioni di acquisto del sito (e.g. carrello, pagamento etc.).



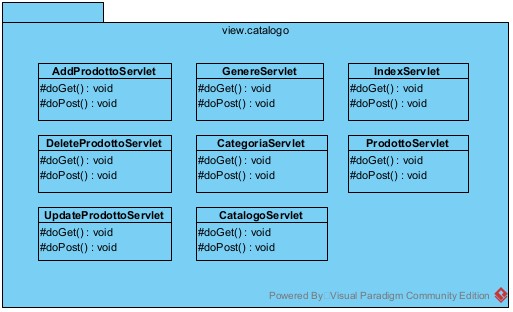
### 3.1.3 view.utente

Questo sottopacchetto contiene le Servlet adibite alle funzioni di gestione degli account nonché di autenticazione al sistema.



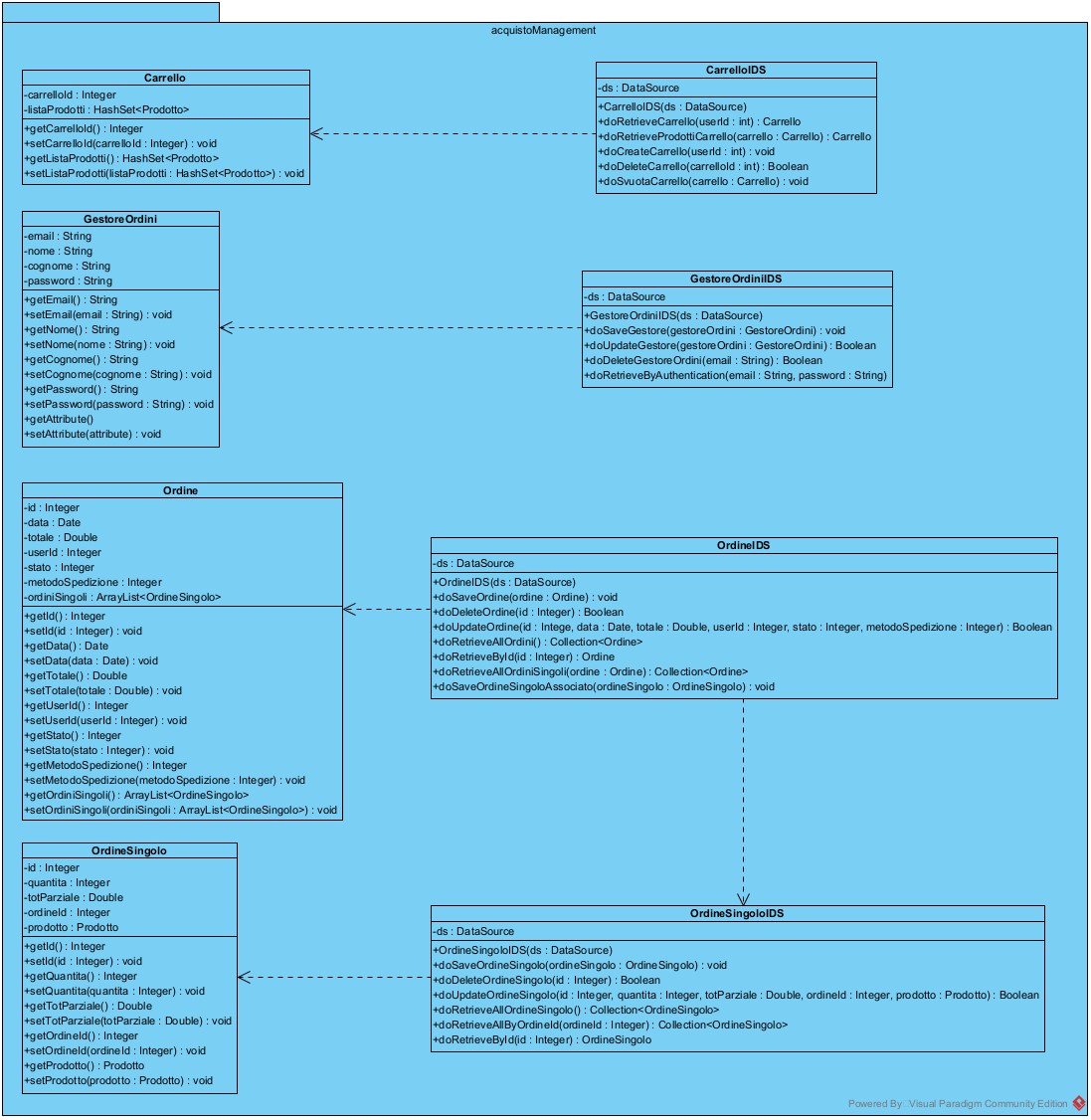
### 3.1.4 view.catalogo

Questo sottopacchetto contiene le Servlet e adibite alle funzioni di gestione del catalogo (e.g. visualizzazione catalogo).



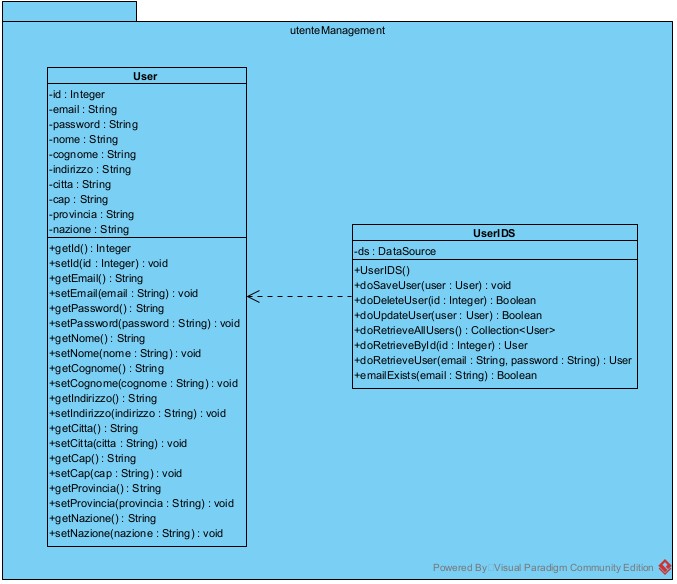
## *3.2 acquistoManagement*

Questo pacchetto contiene le classi Java (Bean e DAO) adibite alle funzioni di acquisto (e.g. creazione di un nuovo ordine nel DB).



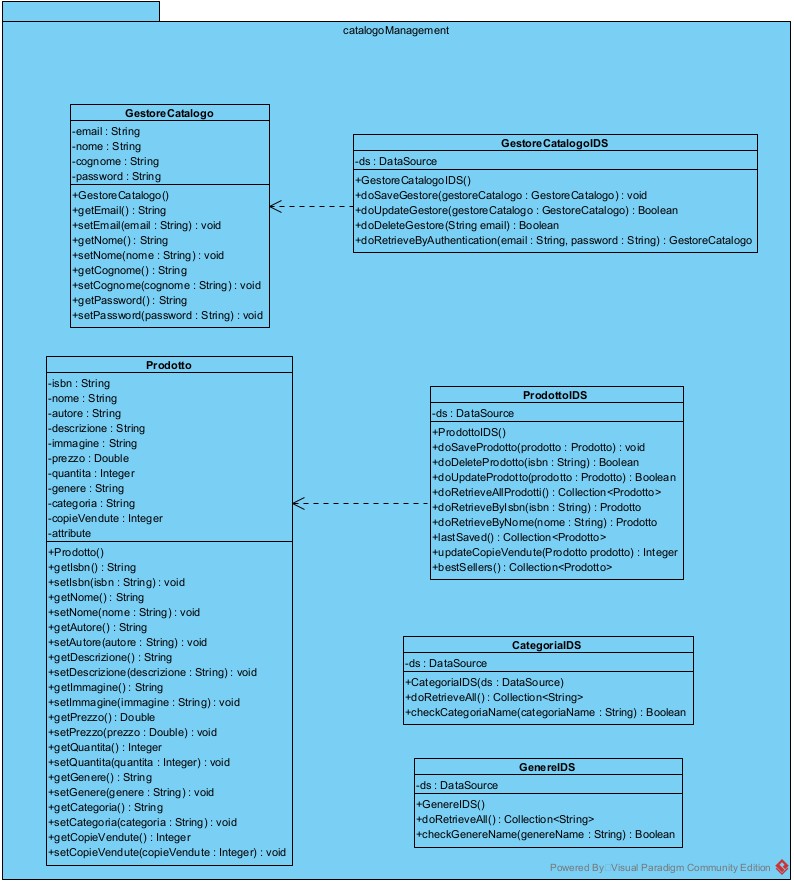
## *3.3 utenteManagement*

Questo pacchetto contiene le classi Java (Bean e DAO) adibite alle funzioni di gestione degli account e delle loro informazioni.



## *3.4 catalogoManagement*

Questo pacchetto contiene le classi Java (Bean e DAO) adibite alle funzioni di gestione del catalogo (tutte le informazioni sui prodotti, su dove sono immagazzinati etc.).



# 4. Interfacce di classe

* 1. *CarrelloIDS*

|  |  |
| --- | --- |
| **CarrelloIDS** | |
| **Descrizione** | Questa classe permette di interfacciarci al DBMS relazione e, in particolare, di accedere, modificare e interrogare l’entità Carrello. |
| **Pre-condizione** | context CarrelloIDS::doCreateCarrello(userId: Integer)  pre: userId <> null and userId > 0  context CarrelloIDS::doDeleteCarrello(carrelloId: Integer)  pre: carrelloId <> null and carrelloId > 0  context CarrelloIDS::doRetrieveCarrello(userId: Integer)  pre: userId <> null and userId > 0  context CarrelloIDS::doRetrieveProdottiCarrello(carrello: Carrello)  pre: carrello <> null  context CarrelloIDS::doSvuotaCarrello(carrello: Carrello)  pre: carrello <> null and not carrello.listaProdotti.isEmpty() |
| **Post-condizione** | context CarrelloIDS::doCreateCarrello(userId: Integer)  post:  Carrello.allInstances()->exists(c | c.userId = userId)  and  Prodotto.allInstances()->forAll(p | CarrelloProdotto.allInstances()->exists(cp | cp.prodotto = p and cp.carrello.userId = userId))  context CarrelloIDS::doDeleteCarrello(carrelloId: Integer)  post:  not Carrello.allInstances()->exists(c | c.id = carrelloId)  and  not CarrelloProdotto.allInstances()->exists(cp | cp.carrello.id = carrelloId)  and  result = (Carrello.allInstances()->forAll(c | c.id <> carrelloId) and CarrelloProdotto.allInstances()->forAll(cp | cp.carrello.id <> carrelloId))  context CarrelloIDS::doRetrieveCarrello(userId: Integer)  post:  Carrello.allInstances()->exists(c | c.userId = userId)  and  (if Carrello.allInstances()->exists(c | c.userId = userId) then result.userId = userId else result.listaProdotti->isEmpty() endif)  context CarrelloIDS::doRetrieveProdottiCarrello(carrello: Carrello)  post:  result = carrello  and  (if not carrello.listaProdotti->isEmpty() then result.listaProdotti->notEmpty() else result.listaProdotti->isEmpty() endif)  context CarrelloIDS::doSvuotaCarrello(carrello: Carrello)  post:  Carrello.allInstances()->exists(c | c = carrello and c.listaProdotti->notEmpty())  and  Prodotto.allInstances()->forAll(p | CarrelloProdotto.allInstances()->exists(cp | cp.prodotto = p and cp.carrello = carrello)) |
| **Invariante** |  |

* 1. *GestoreOrdiniIDS*

|  |  |
| --- | --- |
| **GestoreOrdiniIDS** | |
| **Descrizione** | Questa classe permette di interfacciarci al DBMS relazione e, in particolare, di accedere, modificare e interrogare l’entità GestoreOrdini. |
| **Pre-condizione** | context GestoreOrdiniIDS::doSaveGestore(gestoreOrdini: GestoreOrdini)  pre:  gestoreOrdini <> null  and gestoreOrdini.nome <> null and gestoreOrdini.nome <> ""  and gestoreOrdini.cognome <> null and gestoreOrdini.cognome <> ""  and gestoreOrdini.email <> null and gestoreOrdini.email <> ""  and gestoreOrdini.password <> null and gestoreOrdini.password <> ""  context GestoreOrdiniIDS::doUpdateGestore(gestoreOrdini: GestoreOrdini)  pre:  gestoreOrdini <> null  and gestoreOrdini.nome <> null and gestoreOrdini.cognome <> null  and gestoreOrdini.email <> null and gestoreOrdini.email <> ""  and gestoreOrdini.password <> null and gestoreOrdini.password <> ""  context GestoreOrdiniIDS::doDeleteGestore(email: String)  pre:  email <> null and email <> ""  context GestoreOrdiniIDS::doRetrieveByAuthentication(email: String, password: String)  pre:  email <> null and email <> ""  and password <> null and password <> "" |
| **Post-condizione** | context GestoreOrdiniIDS::doSaveGestore(gestoreOrdini: GestoreOrdini)  post:  GestoreOrdini.allInstances()->exists(go | go = gestoreOrdini)  context GestoreOrdiniIDS::doUpdateGestore(gestoreOrdini: GestoreOrdini)  post:  GestoreOrdini.allInstances()->exists(go | go.email = gestoreOrdini.email)  and  result = (GestoreOrdini.allInstances()->forAll(go | go.email <> gestoreOrdini.email) and GestoreOrdini.allInstances()->exists(go | go = gestoreOrdini))  context GestoreOrdiniIDS::doDeleteGestore(email: String)  post:  result = (GestoreOrdini.allInstances()->forAll(go | go.email <> email) and GestoreOrdini.allInstances()->exists(go | go.email = email))  context GestoreOrdiniIDS::doRetrieveByAuthentication(email: String, password: String)  post:  (GestoreOrdini.allInstances()->exists(go | go.email = email and go.password = password) implies result = GestoreOrdini.allInstances()->any(go | go.email = email and go.password = password))  and  (not GestoreOrdini.allInstances()->exists(go | go.email = email and go.password = password) implies result = null) |
| **Invariante** |  |

* 1. *OrdineIDS*

|  |  |
| --- | --- |
| **OrdineIDS** | |
| **Descrizione** | Questa classe permette di interfacciarci al DBMS relazione e, in particolare, di accedere, modificare e interrogare l’entità Ordine. |
| **Pre-condizione** | context OrdineIDS::doSaveOrdine(ordine: Ordine)  pre:  ordine <> null  and ordine.id > 0  and ordine.data <> null  and ordine.totale > 0.0  and ordine.userId > 0  and ordine.stato >= 1  and ordine.stato <= 3  and ordine.metodoSpedizione >= 1  and ordine.metodoSpedizione <= 3  and not ordine.ordiniSingoli->isEmpty()  context OrdineIDS::doDeleteOrdine(id: Integer)  pre:  id > 0  context OrdineIDS::doUpdateOrdine(id: Integer, data: Date, totale: Double, userId: Integer, stato: Integer, metodoSpedizione: Integer)  pre:  id > 0  and data <> null  and totale > 0.0  and userId > 0  and stato >= 1  and stato <= 3  and metodoSpedizione >= 1  and metodoSpedizione <= 3  context OrdineIDS::doRetrieveAllOrdini()  pre:  true  context OrdineIDS::doRetrieveById(id: Integer)  pre:  id > 0  context OrdineIDS::doRetrieveAllOrdiniSingoli(ordine: Ordine)  pre:  ordine <> null  and ordine.id > 0  context OrdineIDS::doSaveOrdineSingoloAssociato(ordineSingolo: OrdineSingolo)  pre:  ordineSingolo <> null  and ordineSingolo.id > 0 |
| **Post-condizione** | context OrdineIDS::doSaveOrdine(ordine: Ordine)  post:  Ordine.allInstances()->exists(o | o = ordine and ordine.ordiniSingoli->  forAll(os | OrdineSingolo.allInstances()->exists(osi | osi = os)))  context OrdineIDS::doDeleteOrdine(id: Integer)  post:  result = (Ordine.allInstances()->forAll(o | o.id <> id) and not Ordine.allInstances()->exists(o | o.id = id))  context OrdineIDS::doUpdateOrdine(id: Integer, data: Date, totale: Double, userId: Integer, stato: Integer, metodoSpedizione: Integer)  post:  result = (Ordine.allInstances()->exists(o | o.id = id and o.data = data and o.totale = totale and o.userId = userId and o.stato = stato and o.metodoSpedizione = metodoSpedizione))  context OrdineIDS::doRetrieveAllOrdini()  post:  result = Ordine.allInstances()  context OrdineIDS::doRetrieveById(id: Integer)  post:  (Ordine.allInstances()->exists(o | o.id = id) implies result = Ordine.allInstances()->any(o | o.id = id))  and  (not Ordine.allInstances()->exists(o | o.id = id) implies result = null)  context OrdineIDS::doRetrieveAllOrdiniSingoli(ordine: Ordine)  post:  result = ordine.ordiniSingoli  context OrdineIDS::doSaveOrdineSingoloAssociato(ordineSingolo: OrdineSingolo)  post:  OrdineSingolo.allInstances()->exists(os | os = ordineSingolo) |
| **Invariante** |  |

* 1. *OrdineSingoloIDS*

|  |  |
| --- | --- |
| **OrdineSingoloIDS** | |
| **Descrizione** | Questa classe permette di interfacciarci al DBMS relazione e, in particolare, di accedere, modificare e interrogare l’entità OrdineSingolo. |
| **Pre-condizione** | context OrdineSingoloIDS::doSaveOrdineSingolo(ordineSingolo: OrdineSingolo)  pre:  ordineSingolo <> null  and ordineSingolo.id > 0  and ordineSingolo.quantita > 0  and ordineSingolo.totParziale > 0.0  and ordineSingolo.ordineId > 0  and ordineSingolo.prodotto <> null  context OrdineSingoloIDS::doDeleteOrdineSingolo(id: Integer)  pre:  id > 0  context OrdineSingoloIDS::doUpdateOrdineSingolo(id: Integer, quantita: Integer, totParziale: Double, prodotto: Prodotto)  pre:  id > 0  and quantita > 0  and totParziale > 0.0  and ordineId > 0  and prodotto <> null  context OrdineSingoloIDS::doRetrieveAllOrdineSingolo()  pre:  true  context OrdineSingoloIDS::doRetrieveAllByOrdineId(ordineId: Integer)  pre:  ordineId > 0  context OrdineSingoloIDS::doRetrieveById(id: Integer)  pre:  id > 0 |
| **Post-condizione** | context OrdineSingoloIDS::doSaveOrdineSingolo(ordineSingolo: OrdineSingolo)  post:  OrdineSingolo.allInstances()->exists(os | os = ordineSingolo)  context OrdineSingoloIDS::doDeleteOrdineSingolo(id: Integer)  post:  result = (OrdineSingolo.allInstances()->forAll(os | os.id <> id) and not OrdineSingolo.allInstances()->exists(os | os.id = id))  context OrdineSingoloIDS::doUpdateOrdineSingolo(id: Integer, quantita: Integer, totParziale: Double, prodotto: Prodotto)  post:  result = (OrdineSingolo.allInstances()->exists(os | os.id = id and os.quantita = quantita and os.totParziale = totParziale and os.prodotto = prodotto))  context OrdineIDS::doRetrieveAllOrdineSingolo()  post:  result = OrdineSingolo.allInstances()  context OrdineIDS::doRetrieveAllByOrdineId(ordineId: Integer)  post:  result = OrdineSingolo.allInstances()->select(os | os.ordineId = ordineId)  context OrdineIDS::doRetrieveById(id: Integer)  post:  (OrdineSingolo.allInstances()->exists(os | os.id = id) implies result = OrdineSingolo.allInstances()->any(os | os.id = id))  and  (not OrdineSingolo.allInstances()->exists(os | os.id = id) implies result = null) |
| **Invariante** |  |

* 1. *CategoriaIDS*

|  |  |
| --- | --- |
| **CategoriaIDS** | |
| **Descrizione** | Questa classe permette di interfacciarci al DBMS relazione e, in particolare, di accedere, modificare e interrogare l’entità Categoria. |
| **Pre-condizione** | **context** CategoriaIDS:: doRetrieveAll()  **pre**:  **context** CategoriaIDS:: checkCategoriaName(String categoriaName)  **pre**: categoriaName != null |
| **Post-condizione** | **context** CategoriaIDS:: doRetrieveAll()  **post**: restituisce la lista di tutte le categorie presenti nel Database  **context** CategoriaIDS:: checkCategoriaName(String categoriaName)  **post**: controlla che la stringa categoriaName corrisponda ad una delle categorie nel Database. Restituisce true se l’operazione è andata a buon fine, altrimenti false |
| **Invariante** |  |

* 1. *GenereIDS*

|  |  |
| --- | --- |
| **GenereIDS** | |
| **Descrizione** | Questa classe permette di interfacciarci al DBMS relazione e, in particolare, di accedere, modificare e interrogare l’entità Genere. |
| **Pre-condizione** | context GenereIDS::doRetrieveAll()  pre:  true  context GenereIDS::checkGenereName(genereName: String)  pre:  genereName <> null |
| **Post-condizione** | context GenereIDS::doRetrieveAll()  post:  result = Genere.allInstances()  context GenereIDS::checkGenereName(genereName: String)  post:  result = Genere.allInstances()->exists(g | g.nome = genereName) |
| **Invariante** |  |

* 1. *GestoreCatalogoIDS*

|  |  |
| --- | --- |
| **GestoreCatalogoIDS** | |
| **Descrizione** | Questa classe permette di interfacciarci al DBMS relazione e, in particolare, di accedere, modificare e interrogare l’entità GestoreCatalogo. |
| **Pre-condizione** | context GestoreCatalogoIDS::doSaveGestore(gestoreCatalogo: GestoreCatalogo)  pre:  gestoreCatalogo <> null  and gestoreCatalogo.nome <> null and gestoreCatalogo.nome <> ""  and gestoreCatalogo.cognome <> null and gestoreCatalogo.cognome <> ""  and gestoreCatalogo.email <> null and gestoreCatalogo.email <> ""  and gestoreCatalogo.password <> null and gestoreCatalogo.password <> ""  context GestoreCatalogoIDS::doUpdateGestore(gestoreCatalogo: GestoreCatalogo)  pre:  gestoreCatalogo <> null  and gestoreCatalogo.nome <> null and gestoreCatalogo.cognome <> null  and gestoreCatalogo.email <> null and gestoreCatalogo.email <> ""  and gestoreCatalogo.password <> null and gestoreCatalogo.password <> ""  context GestoreCatalogoIDS::doDeleteGestore(email: String)  pre:  email <> null and email <> ""  context GestoreCatalogoIDS::doRetrieveByAuthentication(email: String, password: String)  pre:  email <> null and email <> ""  and password <> null and password <> "" |
| **Post-condizione** | context GestoreCatalogoIDS::doSaveGestore(gestoreCatalogo: GestoreCatalogo)  post:  GestoreCatalogo.allInstances()->exists(gc | gc = gestoreCatalogo)  context GestoreCatalogoIDS::doUpdateGestore(gestoreCatalogo: GestoreCatalogo)  post:  GestoreCatalogo.allInstances()->exists(gc | gc.email = gestoreCatalogo.email)  and  result = (GestoreCatalogo.allInstances()->forAll(gc | gc.email <> gestoreCatalogo.email) and GestoreCatalogo.allInstances()->exists(gc | gc = gestoreCatalogo))  context GestoreCatalogoIDS::doDeleteGestore(email: String)  post:  result = (GestoreCatalogo.allInstances()->forAll(gc | gc.email <> email) and GestoreCatalogo.allInstances()->exists(gc | gc.email = email))  context GestoreCatalogoIDS::doRetrieveByAuthentication(email: String, password: String)  post:  (GestoreCatalogo.allInstances()->exists(gc | gc.email = email and gc.password = password) implies result = GestoreCatalogo.allInstances()->any(gc | gc.email = email and gc.password = password))  and  (not GestoreCatalogo.allInstances()->exists(gc | gc.email = email and gc.password = password) implies result = null) |
| **Invariante** |  |

* 1. *ProdottoIDS*

|  |  |
| --- | --- |
| **ProdottoIDS** | |
| **Descrizione** | Questa classe permette di interfacciarci al DBMS relazione e, in particolare, di accedere, modificare e interrogare l’entità Prodotto. |
| **Pre-condizione** | context ProdottoIDS::doSaveProdotto(prodotto: Prodotto)  pre:  prodotto <> null  and prodotto.isbn.size() = 13  and prodotto.nome <> null and prodotto.nome <> ""  and prodotto.autore <> null and prodotto.autore <> ""  and prodotto.descrizione <> null and prodotto.descrizione <> ""  and prodotto.immagine <> null and prodotto.immagine <> ""  and prodotto.prezzo > 0 and prodotto.quantita > 0  and prodotto.genere <> null and prodotto.genere <> ""  and prodotto.categoria <> null and prodotto.categoria <> ""  context ProdottoIDS::doDeleteProdotto(isbn: String)  pre:  isbn <> null and isbn.size() = 13  context ProdottoIDS::doUpdateProdotto(prodotto: Prodotto)  pre:  prodotto <> null  and prodotto.nome <> null  and prodotto.autore <> null  and prodotto.descrizione <> null  and prodotto.immagine <> null  and prodotto.prezzo <> null  and prodotto.quantita <> null  and prodotto.genere <> null  and prodotto.categoria <> null  context ProdottoIDS::doRetrieveAllProdotti()  pre:  true  context ProdottoIDS::doRetrieveByIsbn(isbn: String)  pre:  isbn <> null and isbn.size() = 13  context ProdottoIDS::doRetrieveByNome(nome: String)  pre:  nome <> null  context ProdottoIDS::lastSaved()  pre:  true  context ProdottoIDS::updateCopieVendute(prodotto: Prodotto)  pre:  prodotto <> null and prodotto.copieVendute <> null  context ProdottoIDS::bestSellers()  pre:  true |
| **Post-condizione** | context ProdottoIDS::doSaveProdotto(prodotto: Prodotto)  post:  Prodotto.allInstances()->exists(p | p = prodotto)  context ProdottoIDS::doDeleteProdotto(isbn: String)  post:  result = (Prodotto.allInstances()->forAll(p | p.isbn <> isbn) and Prodotto.allInstances()->exists(p | p.isbn = isbn))  context ProdottoIDS::doUpdateProdotto(prodotto: Prodotto)  post:  result = (Prodotto.allInstances()->exists(p | p.isbn = prodotto.isbn and p = prodotto))  context ProdottoIDS::doRetrieveAllProdotti()  post:  result = Prodotto.allInstances()  context ProdottoIDS::doRetrieveByIsbn(isbn: String)  post:  (Prodotto.allInstances()->exists(p | p.isbn = isbn) implies result = Prodotto.allInstances()->any(p | p.isbn = isbn))  and  (not Prodotto.allInstances()->exists(p | p.isbn = isbn) implies result = null)  context ProdottoIDS::doRetrieveByNome(nome: String)  post:  (Prodotto.allInstances()->exists(p | p.nome = nome) implies result = Prodotto.allInstances()->any(p | p.nome = nome))  and  (not Prodotto.allInstances()->exists(p | p.nome = nome) implies result = null)  context ProdottoIDS::lastSaved()  post:  result = Prodotto.allInstances()->sortedBy(dataCreazione)->firstN(5)  context ProdottoIDS::updateCopieVendute(prodotto: Prodotto)  post:  result = (Prodotto.allInstances()->exists(p | p.isbn = prodotto.isbn and p.copieVendute = prodotto.copieVendute))  context ProdottoIDS::bestSellers()  post:  result = Prodotto.allInstances()->sortedBy(copieVendute)->lastN(5) |
| **Invariante** |  |

* 1. *UserIDS*

|  |  |
| --- | --- |
| **UserIDS** | |
| **Descrizione** | Questa classe permette di interfacciarci al DBMS relazione e, in particolare, di accedere, modificare e interrogare l’entità *site\_user*. |
| **Pre-condizione** | context UserIDS::doSaveUser(user: User)  pre:  user <> null  and user.id > 0  and user.email <> null and user.email <> ""  and user.password <> null and user.password <> ""  and user.nome <> null and user.nome <> ""  and user.cognome <> null and user.cognome <> ""  and user.indirizzo <> null and user.indirizzo <> ""  and user.citta <> null and user.citta <> ""  and user.cap <> null and user.cap.size() = 5  and user.provincia <> null and user.provincia.size() = 2  and user.nazione <> null and user.nazione <> ""  context UserIDS::doDeleteUser(id: String)  pre:  id > 0  context UserIDS::doUpdateUser(user: User)  pre:  user <> null  and user.id > 0  and user.email <> null  and user.password <> null  and user.nome <> null  and user.cognome <> null  and user.indirizzo <> null  and user.citta <> null  and user.cap <> null and user.cap.size() = 5  and user.provincia <> null and user.provincia.size() = 2  and user.nazione <> null  context UserIDS::doRetrieveAllUsers()  pre:  true  context UserIDS::doRetrieveById(id: Integer)  pre:  id > 0  context UserIDS::doRetrieveUser(email: String, password: String)  pre:  email <> null and password <> null  context UserIDS::emailExists(email: String)  pre:  email <> null |
| **Post-condizione** | context UserIDS::doSaveUser(user: User)  post:  User.allInstances()->exists(u | u = user)  context UserIDS::doDeleteUser(id: String)  post:  result = (User.allInstances()->forAll(u | u.id <> id) and  User.allInstances()->exists(u | u.id = id))  context UserIDS::doUpdateUser(user: User)  post:  result = (User.allInstances()->exists(u | u.id = user.id and u = user))  context UserIDS::doRetrieveAllUsers()  post:  result = User.allInstances()  context UserIDS::doRetrieveById(id: Integer)  post:  (User.allInstances()->exists(u | u.id = id) implies result =  User.allInstances()->any(u | u.id = id))  and  (not User.allInstances()->exists(u | u.id = id) implies result = null)  context UserIDS::doRetrieveUser(email: String, password: String)  post:  (User.allInstances()->exists(u | u.email = email and u.password = password) implies result = User.allInstances()->any(u | u.email = email and u.password = password))  and  (not User.allInstances()->exists(u | u.email = email and u.password = password) implies result = null)  context UserIDS::emailExists(email: String)  pre:  result = User.allInstances()->exists(u | u.email = email) |
| **Invariante** |  |