OBJECT DESIGN DOCUMENT

Studenti

Bene Sabato 05121**06213**

Cozzolino Lidia 05121**08934**

Napoli Riccardo 05121**09152**

Penna Alessandro 05121**06225**

NETPHARMA
La tua farmacia online

OBJECT DESIGN DOCUMENT

Object design document

1. Introduzione

Il seguente documento si pone come obbiettivo quello di descrivere le linee guida in fase di implementazione del sistema NetPharma. In questo documento verranno descritti i trade-off generali e le convenzioni per la nomenclatura ed implementazione di classi ed interfacce.

1.1) Object Design Trade-offs

Interfaccia vs Usabilità:

L'interfaccia deve essere realizzata in maniera chiara e semplice in modo da poter essere utilizzata da diversi tipi di utenti, facendo uso di semplici e ben visibili form e pulsanti con descrizioni esplicative.

Comprensibilità vs Tempo:

Essendo i tempi di sviluppo del sistema limitati, il codice NON sarà accompagnato da commenti che ne semplificherebbero la comprensione, per permettere l'implementazione del sistema in tempi più contenuti. Di conseguenza, la fase di testing ed eventuali future modifiche saranno più complesse.

Response Time vs Hardware:

Il sistema deve garantire tempi di risposta rapidi nonostante alto carico di utenza, per cui l'hardware utilizzato dovrà essere adeguato per poterlo permettere.

Prestazioni vs Costi:

Lo sviluppo del sistema non prevede l'utilizzo di librerie o componenti a pagamento, essendo il progetto privo di budget economico.

Sicurezza vs Efficienza:

Dati i tempi piuttosto ristretti per lo sviluppo, ci limiteremo ad implementare sistemi di sicurezza basati su username e password crittografata, garantendo un controllo sugli accessi basato su determinati ruoli.

1.2) Linee Guida per la Documentazione delle Interfacce

Di seguito le linee guida che gli sviluppatori rispetteranno durante l'implementazione dell'interfaccia:

Naming Conventions:

Per la nomenclatura è necessario utilizzare nomi di lunghezza medio-corta, che non siano abbreviati e che contengano solo caratteri alfanumerici.

I nomi delle variabili e dei metodi devono seguire la cosiddetta "dromedaryCase" (iniziare con una lettera minuscola e le parole successive con una lettera maiuscola).

Per i metodi in particolare è necessario, salvo particolari casi, definirne il nome con un verbo che specifica l'azione del metodo seguito dall'oggetto su cui l'azione si verifica.

I metodi d'accesso e di modifica delle variabili devono sempre iniziare rispettivamente con get/set prima del nome della variabile.

I commenti, ove utilizzati, devono essere scritti sulla stessa riga per le variabili e prima della dichiarazione per i metodi, descrivendone l'utilizzo.

I nomi delle classi e delle interfacce devono seguire la cosiddetta "CamelCase" (iniziare ogni parola con una lettera maiuscola).

1.3) Definizioni, acronimi e abbreviazioni

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

1.4) Riferimenti

Requirements Analysis Document (RAD). System Design Document (SDD).

2. Packages

Il nostro sistema è diviso in tre livelli (architettura three-tier MVC)

- Presentation layer
- Application layer
- Storage layer

Il package NetPharma contiene sottopackage che a loro volta contengono classi dedicate.

- Presentation layer:

Rappresenta la parte del sistema che si occupa dell'interfaccia utente, contiene quindi le risorse atte ad interagire con l'utente e contiene i form di input ed output e le servlet.

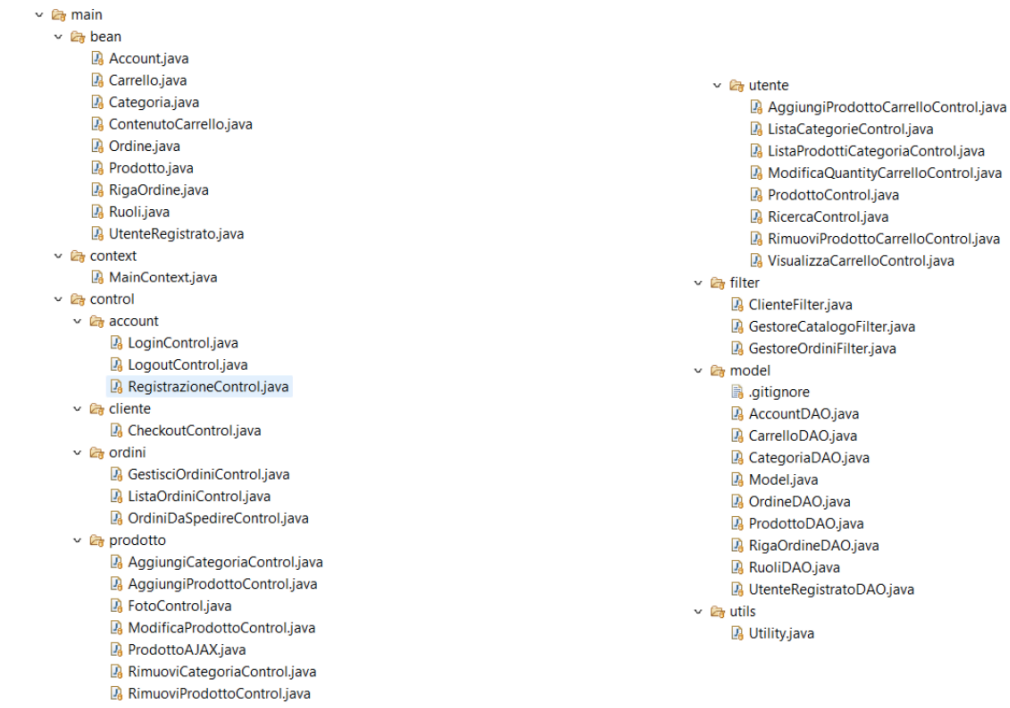
- Application layer

Contiene DAO e Bean, cioè le risorse atte ad interfacciarsi con il database e a modellare i dati.

- Storage layer

Consiste in un dbms che ha il compito di gestire il database per memorizzare, aggiornare e cancellare i dati dell'applicazione nonché di restituirli e ricevere le richieste dall'application layer.

Struttura Packages :



3. Class Interfaces

Account
- id : int - username : String - password : String - orderCount : int
+ getId( ) : int + getUsername( ) : String + getPassword( ) : String + getOrderCount( ) : int + setId(id : int) : void + setUsername(username : String) : void + setPassword(password : String) : void + setOrderCount(orderCount : int) : void

context Account::getId( )
post: result = id

context Account::getUsername( )
post: result = username

context Account::getPassword( )
post: result = password

context Account::getOrderCount( )
post: result = orderCount

context Account::setId(id)
post: self.id = id

context Account::setUsername(username)
post: self.username = username

context Account::setPassword(password)
post: self.password = password

context Account::setOrderCount(orderCount)
post: self.orderCount = orderCount

AccountModel
//
+ authenticate(username : String, password : String) : Account + checkUsername(username : String) : boolean + register(account : Account, user : UtenteRegistrato, r : Ruoli) : boolean + updateOrderCount(accountID : int, newOrderCount : int) : boolean

context AccountModel::authenticate(username, password)
post: result = database.Account -> select(a | a.username = username and a.password = password)

context AccountModel::checkUsername(username)
post: result = database.Account -> exists(a | a.username = username)

context AccountModel::register(account, user, r)
pre: account.username != null and account.password != null and account.order_count != null and not(database.Account -> exists(a | a.username = account.username)) and user.genere != null and user.nome != null and user.cognome != null and user.email != null and user.nascita != null and user.account = account.id
and not(database.UtenteRegistrato -> exists(u | u.email = user.email)) and r != null
post: result = (database.Account -> exists(a | a.username = account.username
and a.password = account.password and a.order_count = account.order_count) and dataBase.UtenteRegistrato -> exists(u | u.genere = user.genere and u.nome = user.nome and u.cognome = user.cognome and u.email = user.email and u.nascita = user.nascita and u.account = account.id) and r.ruoli -> forAll(ruolo | database.Ruoli ->exists(r | r.account = account.id and r.ruolo = ruolo)))

context AccountModel::updateOrderCount(accountID, newOrderCount)
post: result = database.Account->exists(a | a.id = accountId and a.orderCount = newOrderCount)

Ruoli
+ Ruolo {CL, GO, GC} : enum - account : int - ruoli : ArrayList<Ruolo>
+ getAccount( ) : int + getRuoli( ) : ArrayList<Ruolo> + setAccount(account : int) : void + setRuoli(ruoli : ArrayList<Ruolo>) : void + addRuolo(ruolo : Ruolo) : void + removeRuolo(ruolo : Ruolo) : void + stringToRole(role : String) : Ruolo + roleToString(role : Ruolo) : String

context Ruoli::getAccount( )
post: result = account

context Ruoli::getRuoli( )
post: result = ruoli

context Ruoli::setAccount(account)
post: self.account = account

context Ruoli::setRuoli(ruoli)
post: self.ruoli = ruoli

context Ruoli::addRuolo(ruolo)
post : ruoli->exists(r | r = ruolo)

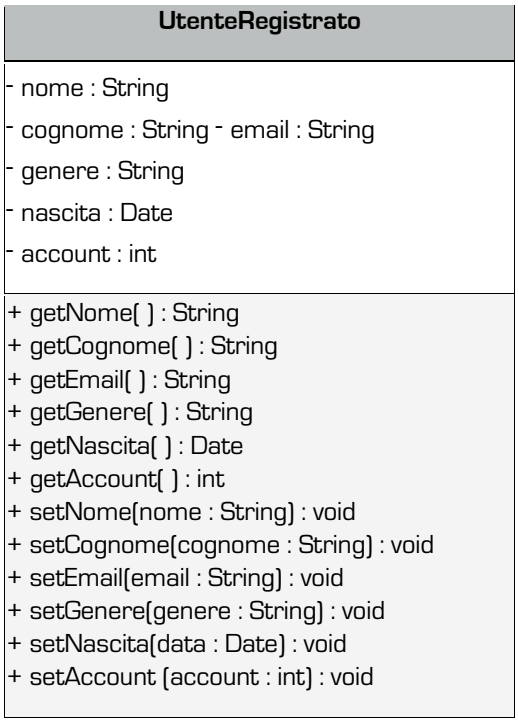
context Ruoli::removeRuolo(ruolo)
post : not(ruoli->exists(r | r = ruolo))

context Ruoli::stringToRole(ruolo)
post : result = Ruolo corrispondente a ruolo

context Ruoli::roleToString(ruolo)
post : result = String corrispondente a ruolo

RuoliModel
//
+ doRetrieveByAccount(account : int) : Ruoli

context RuoliModel::doRetriveByAccount(account)
post: result.getRuoli( ) = database.Ruoli -> select(r | r.account = account) and result.account = account



context UtenteRegistrato::getNome( )
post: result = nome

context UtenteRegistrato::getCognome( )
post: result = cognome

context UtenteRegistrato::getEmail( )
post: result = email

context UtenteRegistrato::getGenere( )
post: result = genere

context UtenteRegistrato::getNascita( )
post: result = nascita

context UtenteRegistrato::getAccount( )
post: result = account

context UtenteRegistrato::setNome(nome)
post: self.nome = nome

context UtenteRegistrato::setCognome(cognome)
post: self.cognome = cognome

context UtenteRegistrato::setEmail(email)
post: self.email = email

context UtenteRegistrato::setGenere(genere)
post: self.genere = genere

context UtenteRegistrato::setNascita(nascita)
post: self.nascita = nascita

context UtenteRegistrato::setAccount(account)
post: self.ruolo = ruolo

UtenteRegistratoModel
//
+ doRetrieveByAccount(account : int) : UtenteRegistrato + checkEmail(email : String) : boolean

context UtenteRegistratoModel::doRetrieveByAccount(account)
post: result = database.UtenteRegistrato -> select(user | user.account = account)

context UtenteRegistratoModel::checkEmail(email)
post: result = database.UtenteRegistrato -> exists(user | user.email = email)

ContenutoCarrello
- prodotto : Prodotto - quantity : int
+ getProdotto( ) : String + getQuantity( ) : int + setProdottol(prodotto : Prodotto) : void + setQuantity(quantità : int) : void

context CartItem::getProdotto( )
post: result = prodotto

context CartItem::getQuantity( )
post: result = quantity

context CartItem::setProdotto(prodotto)
post: self.prodotto = prodotto

context CartItem::setQuantity(quantity) **pre:** quantity > 0
post: self.quantity = quantity

Carrello
- prodotti : ArrayList<ContenutoCarrello>
+ getItems( ) : ArrayList<ContenutoCarrello> + addItem(p : Prodotto) : void + removeItem(p : Prodotto) : void + setItem(p : Prodotto, quantity : int) + getTotale( ) : float + getNProdotti( ) : int + clearCart ( ) : void

context Carrello::getItems( )
post: result = prodotti

context Carrello::addItem(prodotto)
post: prodotti -> exists(c | c.prodotto.id = prodotto.id and c.quantity = (@pre.c.quantity+1))

context Carrello::removeItem(prodotto)
post: not(prodotti -> exists(c | c.prodotto.id = prodotto.id))

context Carrello::getTotale( )
post: result = sum(cartItem.getProdotto().getPrice( ) | getItems( ) -> includes(cartItem))

context Carrello::getNProdotti( )
post: result = sum(cartItem.getQuantity( ) | getItems( ) -> includes(cartItem))

context Carrello::clearCart( )
post: getItems( ).isEmpty( )

CarrelloModel
//
+ doRetrieveByUsername(username : String) : Carrello + updateQuantity(username : String, prodottoId : int, quantity : int) : boolean + removeProdotto(username : String, prodotto : int) : boolean + insertProdotto(username : String, prodotto : int, quantity : int) : boolean + clearCart(username : String) : boolean

context CarrelloModel::doRetrieveByUsername(username)
post: result = database.Carrello -> select(c | c.cliente = username)

context CarrelloModel::updateQuantity(username, prodottoId, quantity)
pre: quantity > 0 and database.Account -> exists(a | a.username = username)
and database.Prodotto -> exists(p | p.id = prodottoId)
and database.Carrello -> exists(c | c.cliente = username and c.prodotto = prodottoId)
post: database.Carrello -> exists(c | c.cliente = username and c.prodotto = prodottoId and c.quantity = quantity)

context CarrelloModel::removeProdotto(username, prodotto)
post: not(database.Carrello -> exists(c | c.cliente = username and c.prodotto = prodotto))

context CarrelloModel::insertProdotto(username, prodotto, quantity)
quantity > 0 and database.Account -> exists(a | a.username = username)
and database.Prodotto -> exists(p | p.id = prodottoId)
and not(database.Carrello -> exists(c | c.cliente = username and c.prodotto = prodottoId))
post: database.Carrello -> exists(c | c.cliente = username and c.prodotto = prodottoId and c.quantity = quantity)

context CarrelloModel::clearCart(username)
post: not(database.Carrello -> exists(c | c.cliente = username)

Categoria
- int : id - nome : String
+ getId( ) : int + getNome( ) : String + setId(id : int) : void + setNome(nome : String) : void

context Categoria::getId( )
post: result = id

context Categoria::getNome( )
post: result = nome

context Categoria::setId(id)
post: self.id = id

context Categoria::setNome(nome)
post: self.nome = nome

CategoriaModel
//
+ doRetrieveByKey(id : int) : Categoria + doRetrieveByName(nome : String) : Categoria + doRetrieveAll(o : String) Collection<Categoria> + doSave(categoria : Categoria) : void + doDelete(bean : Categoria) : void + checkCategoria(nome : String) : boolean

context CategoriaModel::doRetrieveByKey(id)
post: result = database.Categoria -> select(c | c.id = id)

context CategoriaModel::doRetrieveByName(nome)
post: result = database.Categoria -> select(c | c.nome = nome)

context CategoriaModel::doRetrieveAll(o)
post: result = database.Categoria order by o

context CategoriaModel::doSave(categoria)
pre: categoria.nome != null and not(database.Categoria -> exists(c | c.nome = categoria.nome))
post: database.Categoria -> select(c | c.nome = categoria.nome)

context CategoriaModel::doDelete(bean)
post: not(database.Categoria -> exists(c | c.nome = bean.nome))

context CategoriaModel::checkCategoria(nome)
post: database.Categoria -> exists(c | c.nome = nome)

Utility
//
+ checkNome(nome : String) : Boolean + checkEmail(email : String) : Boolean + checkPassword(password : String) : Boolean

context Utility::checkNome(nome)
post: result = True if isalpha(nome) else False

context Utility::checkEmail(email)
post: result = True if email.match(x@x.x) else False

context Utility::checkPassword(password)
post: result = True if [len(password) >= 8 and containsUppercase(password) and containsLowercase(password) and containsNumber(password) and contains(password, "@ ! # $ % ' - / = ^ \ _ ` { } ~ +")] else False

Prodotto
- id: Int - nome: String - marchio: String - produttore: String – formato: String - descrizione: String – disponibilità: String – prezzo: BigDecimal – categoria: String -foto: Byte[ ]
+ getId( ) : Int + getNome( ) : String + getMarchio( ) : String + getProduttore( ) : String + getFormato( ) : String + getDescrizione( ) : String + getDisponibilità( ) : String + getPrezzo( ) : BigDecimal + getCategoria( ) : String + getFoto( ) : Byte [ ] + setId(id : Int) : void + setNome(nome : String) : void + setMarchio(marchio : String) : void + setProduttore(produttore : String) : void + setFormato(formato : String) : void + setDescrizione(descrizione : String) : void + setDisponibilità(disponibilità : String) : void + setPrezzo(prezzo : BigDecimal) : void + setCategoria(categoria : String) : void + setFoto(foto : byte [ ]) : void

context Prodotto::getId( )
post: result = id

context Prodotto::getNome( )
post: result = nome

context Prodotto::getMarchio( )
post: result = marchio

context Prodotto::getProduttore( )
post: result = produttore

context Prodotto::getFormato( )
post: result = formato

context Prodotto::getDescrizione( )
post: result = descrizione

context Prodotto::getDisponibilità( )
post: result = disponibilità

context Prodotto::getPrezzo( )
post: result = prezzo

context Prodotto::getCategoria( )
post: result = categoria

context Prodotto::getFoto( )
post: result = foto

context Prodotto::setId(id)
post: self.id = id

context Prodotto::setNome(nome)
post: self.nome = nome

context Prodotto::setMarchio(marchio)
post: self.marchio = marchio

context Prodotto::setProduttore(produttore)
post: self.produttore = produttore

context Prodotto::setFormato(formato)
post: self.formato = formato

context Prodotto::setDescrizione(descrizione)
post: self.descrizione = descrizione

context Prodotto::setDisponibilità(disponibilità)
post: self.disponibilità = disponibilità

context Prodotto::setPrezzo(prezzo)
post: self.prezzo = prezzo

context Prodotto::setCategoria(categoria)
post: self.categoria = categoria

context Prodotto::setFoto(foto)
post: self.foto = foto

ProdottoModel
//
+ doRetrieveByKey(id : Int) : Prodotto + doRetrieveAll(order : String) : Collection<Prodotto> + doSave(prodotto : Prodotto) : void + doUpdate(prodotto : Prodotto) : void + doUpdateCategoria(prodottoId : Int, categoria : String) : void + doDelete(prodotto : Prodotto bean) : void + doRetrieveAllByCategoria(categoria : String) : Collection<Prodotto> + doRicerca(nome : String, order : String) : Collection<Prodotto> + doRetrieveSvincolati( ) : Collection<Prodotto>

context ProdottoModel::doRetrieveByKey(id)
pre: id != null
post: result = database.Prodotto -> select (p | prodotto.id = p.id)

context ProdottoModel::doRetrieveAll()
post: result = database.Prodotto

context ProdottoModel::doSave(prodotto)
pre: prodotto.id !=null and prodotto.nome!=null and prodotto.marchio!=null and prodotto.descrizione!=null and prodotto.disponibilità>=null and prodotto.prezzo>=0
post: database.Prodotto -> exists(p | prodotto.id = p.id and prodotto.nome = p.nome and prodotto.marchio = p.marchio and prodotto.produttore= p.produttore and prodotto.formato = p.formato and prodotto.descrizione = p.descrizione and prodotto.disponibilità = p.disponibilità and prodotto.prezzo = p.prezzo and prodotto.categoria = p.categoria and prodotto.foto = p.foto)

context ProdottoModel::doUpdate(prodotto)
pre: prodotto.id !=null and prodotto.nome!=null and prodotto.marchio!=null and prodotto.descrizione!=null and prodotto.disponibilità!=null and prodotto.prezzo!=null and database.Prodotto -> exists(p | p.id = prodotto.id)
post: database.Prodotto -> exists(p | prodotto.id = p.id and prodotto.nome = p.nome and prodotto.marchio = p.marchio and prodotto.produttore= p.produttore and prodotto.formato = p.formato and prodotto.descrizione = p.descrizione and prodotto.disponibilità = p.disponibilità and prodotto.prezzo = p.prezzo and prodotto.categoria = p.categoria and prodotto.foto = p.foto

context ProdottoModel::doUpdateCategoria(prodottoId, categoria)
pre: database.Prodotto -> exists(p | p.id = prodottoId) and database.Categoria -> exists(c | c.nome = categoria and c.id = categoriaId)
post: database.Prodotto -> exists(p | p.id = prodottoId and p.categoria = categoriaId)

context ProdottoModel::doDelete(prodotto)
pre: prodotto.databaseProdotto -> exists(id | prodotto.id = id)
post: not(database.Prodotto -> exists(id | prodotto.id = id))

context ProdottoModel::doRetrieveAllByCategoria(categoria)
post: result **=** database.Prodotto -> select(p | p.categoria = categoria)

context ProdottoModel::doRicerca(nome, order)
pre: nome!=null and order!=null
post: database.Prodotto -> select(p | p.nome contains nome) order by order

context ProdottoModel::doRetrieveSvincolati()
post: result = database.Prodotti -> select(p | p.categoria=null)

Ordine
- nomeRicevente: String - cognomeRicevente: String - email: String - cellulare: String - ncivico: Int - città: String - via: String - paese: String - provincia: String - CAP: String - data_ordine: Date - data_arrivo: Date - id: Int - prezzo: Float - stato: String - cliente: String - righeOrdine: Collection<RigaOrdine>
+ getNomeRicevente: String + getCognomeRicevente: String + getEmail: String + getCellulare: String + getNcivico: Int + getCittà: String + getVia: String + getPaese: String + getProvincia: String + getCAP: String

+ getData_ordine( ) : Date
+ getData_arrivo( ) : Date
+ getId( ) : String
+ getPrezzo( ) : BigDecimal
+ getStato( ) : String
+ getCliente( ) : String

+ getRigaOrdine: Collection<RigaOrdine>
+ setNomeRicevente(nomeRicevente: String) : void
+ setCognomeRicevente(cognomeRicevente: String) : void
+ setEmail(email: String) : void
+ setCellulare(cellulare: String) : void
+ setNcivico(ncivico: Int) : void
+ setCittà(città: String) : void
+ setVia(via: String) : void
+ setPaese(paese: String) : void
+ setProvincia(provincia: String) : void
+ setCAP: String(cap: String) : void
+ setData_ordine(data_ordine : Date) : void
+ setData_arrivo(data_arrivo : Date) : void
+ setId(id : String) : void
+ setPrezzo(prezzo : BigDecimal) : void
+ setStato(stato : String) : void
+ setCliente(Cliente : String) : void
+ setRigaOrdine(rigaOrdine: Collection<RigaOrdine> : void

context Prodotto::getNomeRicevente( )
post: result = nomeRicevente

context Prodotto::getCognomeRicevente( )
post: result = cognomeRicevente

context Prodotto::getEmail( )
post: result = email

context Prodotto::getCellulare( )
post: result = cellulare

context Prodotto::getNcivico( )
post: result = ncivico **c**

ontext Prodotto::getCittà( )
post: result = città

context Prodotto::getVia( )
post: result = via

context Prodotto::getPaese( )
post: result = paese

context Prodotto::getProvincia( )
post: result = provincia

context Prodotto::getCAP( )
post: result = cap

context Prodotto::getData_ordine( )
post: result = data_ordine

context Prodotto::getData_arrivo( )
post: result = data_arrivo

context Prodotto::getId( )
post: result = id

context Prodotto::getPrezzo( )
post: result = prezzo

context Prodotto::getStato( )
post: result = stato

context Prodotto::getCliente( )
post: result = cliente

context Prodotto::getRigaOrdine ( )
post: result = rigaOrdine

context Prodotto::setNomeRicevente(nomeRicevente)
post: self.nomeRicevente = nomeRicevente

context Prodotto::setCognomeRicevente(cognomeRicevente)
post: self.cognomeRicevente = cognomeRicevente

context Prodotto::setEmail(email)
post: self.email = email

context Prodotto::setCellulare(cellulare)
post: self.cellulare = cellulare

context Prodotto::setNcivico(ncivico)
post: self.ncivico = ncivico

context Prodotto::setCittà(città)
post: self.città = città

context Prodotto::setVia(via)
post: self.via = via

context Prodotto::setPaese(paese)
post: self.paese = paese

context Prodotto::setProvincia(provincia)
post: self.provincia = provincia

context Prodotto::setCAP(cap)
post: self.cap = cap

context Prodotto::setData_ordine(data_ordine)
post: self.data_ordine = data_ordine

context Prodotto::setData_arrivo(data_arrivo)
post: self.data_arrivo = data_arrivo

context Prodotto::setId(id)
post: self.id = id

context Prodotto::setPrezzo(prezzo)
post: self.prezzo = prezzo

context Prodotto::setStato(stato)
post: self.stato = stato

context Prodotto::setCliente(cliente)
post: self.cliente = cliente

context Prodotto::setRigaOrdine(rigaOrdine)
post: self.rigaOrdine = rigaOrdine

OrdineModel
//
+ doRetrieveByKey(key : String) : Ordine + doRetrieveAll(order : String) : Collection<Ordine> + doSave(ordine : Ordine bean) : void + doUpdate(ordine : Ordine bean) : void + doDelete(ordine : Ordine bean) : void + doSaveCheck(ordine : Ordine bean) : boolean + doUpdateStatus(ordine : Ordine bean, millis : long) : Boolean + doRetrieveAllDaSpedire() : Collection<Ordine>

context OrdineModel::doRetrieveByKey(key)
pre: key != null
post: result = database.Ordine -> select (o | ordine.key = o.key) **context** OrdineModel::doRetrieveAll()
post: result = database.Ordine

context OrdineModel::doSave(ordine)
pre: ordine.nomeRicevente!=null and ordine.cognomeRicevente!=null and ordine.email!=null and ordine.cellulare!=null and ordine.city!=null and ordine.ncivico!=null and ordine.via!=null and ordine.paese!=null and ordine.provincia!=null and ordine.cap!=null and ordine.cliente!=null and ordine.data_ordine!=null and ordine.stato!=null and ordine.id!=null and ordine.prezzo!=null
post: database.Ordine -> exists(o | ordine.nomeRicevente = o.nomeRicevente and ordine.cognomeRicevente = o.cognomeRicevente and ordine.email = o.email and ordine.cellulare = o.cellulare and ordine.city = o.city and ordine.ncivico = o.ncivico and ordine.via = o.via and ordine.paese = o.paese and ordine.provincia = o.provincia and ordine.cap = o.cap and ordine.cliente = o.cliente and ordine.data_ordine = o.data_ordine and ordine.stato = o.stato and ordine.id = o.id and ordine.prezzo = o.prezzo)

context OrdineModel::doUpdate(ordine)
pre: ordine.nomeRicevente!=null and ordine.cognomeRicevente!=null and ordine.email!=null and ordine.cellulare!=null and ordine.city!=null and ordine.ncivico!=null and ordine.via!=null and ordine.paese!=null and ordine.provincia!=null and ordine.cap!=null and ordine.cliente!=null and ordine.data_ordine!=null and ordine.stato!=null and ordine.id!=null and ordine.prezzo!=null
post: database.Ordine -> exists(o | ordine.nomeRicevente = o.nomeRicevente and ordine.cognomeRicevente = o.cognomeRicevente and ordine.email = o.email and ordine.cellulare = o.cellulare and ordine.city = o.city and ordine.ncivico = o.ncivico and ordine.via = o.via and

ordine.paese = o.paese and ordine.provincia = o.provincia and ordine.cap = o.cap and ordine.cliente = o.cliente and ordine.data_ordine = o.data_ordine and ordine.stato = o.stato and ordine.id = o.id and ordine.prezzo = o.prezzo)

context OrdineModel::doDelete(ordine)
pre: ordine.databaseOrdine -> exists(id | ordine.id = id)
post: not(ordine.databaseOrdine -> exists(id | ordine.id = id))

context OrdineModel::doSaveCheck(ordine)
pre: ordine.nomeRicevente!=null and ordine.cognomeRicevente!=null and ordine.email!=null and ordine.cellulare!=null and ordine.city!=null and ordine.ncivico!=null and ordine.via!=null and ordine.paese!=null and ordine.provincia!=null and ordine.cap!=null and ordine.cliente!=null and ordine.data_ordine!=null and ordine.stato!=null and ordine.id!=null and ordine.prezzo!=null
post: result = database.Ordine -> exists(o | ordine.nomeRicevente = o.nomeRicevente and ordine.cognomeRicevente = o.cognomeRicevente and ordine.email = o.email and ordine.cellulare = o.cellulare and ordine.city = o.city and ordine.ncivico = o.ncivico and ordine.via = o.via and ordine.paese = o.paese and ordine.provincia = o.provincia and ordine.cap = o.cap and ordine.cliente = o.cliente and ordine.data_ordine = o.data_ordine and ordine.stato = o.stato and ordine.id = o.id and ordine.prezzo = o.prezzo))

context OrdineModel::doUpdateStatus(ordine, millis)
pre: ordine.id!=null and ordine.millis>=172800000 and ordine.millis<=864000000
post: : ordine.databaseOrdine -> exists(id | ordine.id = id) and exists(millis | ordine.millis = millis)

context OrdineModel::doRetrieveAllDaSpedire()
post: result = database.Ordine

RigaOrdine
- quantità: Int - prezzo_Al_Prezzo: Int
+ getQuantità( ) : Int + getPrezzo_Al_Pezzo( ) : Int + setQuantità(Quantità : Int) : void + setPrezzo_Al_pezzo(prezzo_al_prezzo : Int) : void

context Prodotto::getQuantità( )
post: result = quantità

context Prodotto::getPrezzo_Al_pezzo( )
post: result = prezzo_al_pezzo

context Prodotto::setQuantità(quantità)
post: self.quantità = quantità

Context Prodotto::setPrezzo_Al_Pezzo(prezzo_al_pezzo)
post: self.prezzo_al_pezzo = prezzo_al_pezzo

RigaOrdineModel
//
+ doRetrieveAllByOrder(orderID : String) : Collection<RigaOrdine> + doSave(rigaOrdine : RigaOrdine bean) : boolean

context RigaOrdineModel::doRetrieveAllByOrder(orderID)
pre: orderID!=null
post: result = database.Ordine

context RigaOrdineModel::doSave(rigaOrdine)
pre: : ordine.prodotto!=null and ordine.ordine!=null and ordine.quantity!=null and ordine.prezzo_al_pezzo!=null
post: database.Ordine -> exists(o | ordine.prodotto = o.prodotto and ordine.ordine = o.ordine and ordine.quantity = o.quantity and ordine.prezzo_al_pezzo = o.prezzo_al_prezzo