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

UtenteRegistrato
- nome : String - cognome : String - email : String - genere : String - nascita : Date - account : int
+ getNome( ) : String + getCognome( ) : String + getEmail( ) : String + getGenere( ) : String + getNascita( ) : Date + getAccount( ) : int + setNome(nome : String) : void + setCognome(cognome : String) : void + setEmail(email : String) : void + setGenere(genere : String) : void + setNascita(data : Date) : void + setAccount (account : int) : void

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)

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))

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