**Università degli Studi di Salerno**
**Corso di Ingegneria del Software**

# Future Forge Gear
# Object Design Document
# Versione 1.1


Data: 22/12/2025

| | |
|---|---|
| Progetto: Future Forge Gear | Versione: 1.1 |
| Documento: Object Design | Data: 29/01/2026 |

**Coordinatore del progetto:**

| Nome | Matricola |
|---|---|
| | |
| | |

**Partecipanti:**

| Nome | Matricola |
|---|---|
| Francesco Mascolo | 0512117352 |
| Francesco Lamanna | 0512117166 |
| Luigi Mazza | 0512118612 |
| | |
| | |
| | |

| | |
|---|---|
| **Scritto da:** | Francesco Lamanna |

# Revision History

| Data | Versione | Descrizione | Autore |
|---|---|---|---|
| 15/12/2025 | 0.1 | Creazione Del documento | Full Team |
| 18/12/2025 | 0.2 | Aggiunta Product Managment e OrderService | Francesco Lamanna |
| 20/12/2025 | 0.3 | Aggiunta RequestService | Francesco Mascolo |
| 22/12/2025 | 1.0 | Revisione del documento | Full Team |
| 29.01/2026 | 1.1 | Piccolissime Modifiche di revisione | Francesco Mascolo |
| | | | |
| | | | |

# Indice

# INTRODUZIONE

Il presente documento di Object Design Document (ODD) descrive le principali scelte di progettazione a livello di oggetti adottate per il sistema Future Forge Gear.
Il documento ha lo scopo di illustrare i trade-off, le linee guida di sviluppo e l'organizzazione del codice.

## Object Design trade-offs

### Robustezza VS Tempo

Per il sistema Future Forge Gear si è scelto di privilegiare una prima forma di robustezza, concentrata principalmente sul controllo degli input e sulla gestione dei casi di errore più significativi. Per raggiungere una forma finale di robustezza avremmo dovuto avere più tempo rispetto a quanto messo a disposizione per la consegna del progetto stesso.

### Attendibilità VS Tempo

L'attendibilità rappresenta un requisito necessario per il corretto funzionamento del sistema. Per questo motivo l'attendibilità è garantita per tutti i metodi messi a disposizione dal sistema durante la sua esecuzione, assicurando un comportamento coerente e affidabile delle funzionalità principali.

### Linee Guida

Al fine di mantenere coerenze e uniformità nello sviluppo del codice, sono state adottate le seguenti convenzioni:

- Le interfacce sono denominate utilizzare il suffisso "-Interface".
- La gestione degli errori avviene tramite eccezioni controllate.
- Le classi responsabili dell'interazione con il database utilizzano il suffisso "Service".
- Le operazioni CRUD sono denominate secondo la convenzione "CrudOperationEntity"
- I metodi di recupero dei dati utilizzano il prefisso "findBy", seguito dal criterio di ricerca.

### Riferimenti

Le scelte progettuali descritte nel presente documento fanno riferimento ai seguenti documenti:

- RAD_FutureForgeGear
- SDD_FutureForgeGear

### Directory

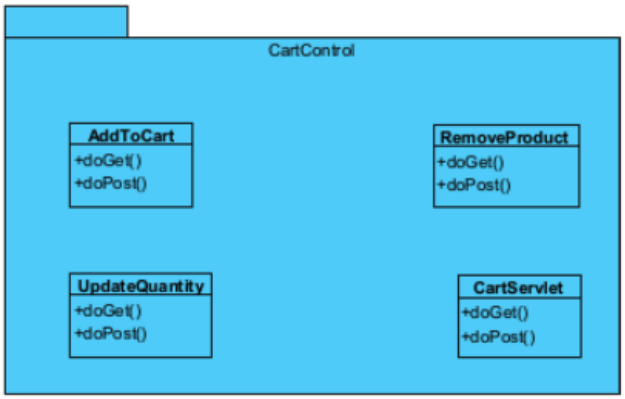
Nella prima versione di implementazione del sistema Future Forge Gear, la struttura del progetto è organizzata in directory e package, ciascuno con un ruolo ben definito.

## Src
La directory src contiene il codice Java necessario all'implementazione del sistema.

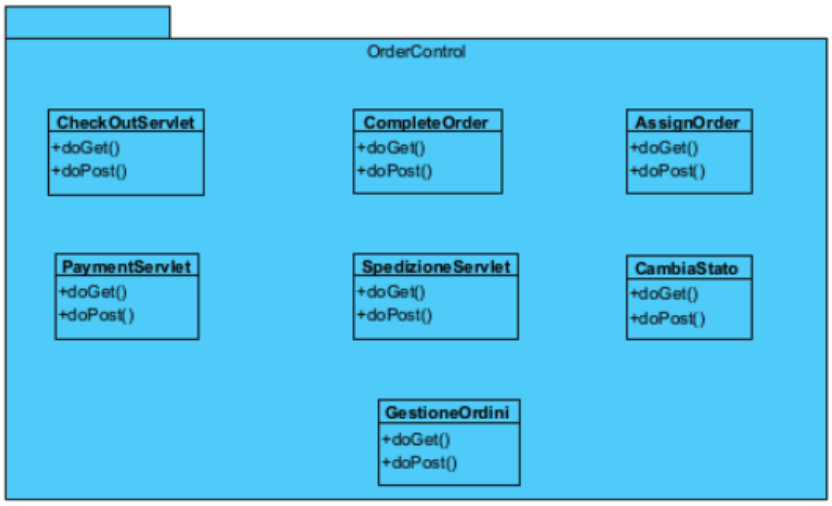
## Database
La directory Database contiene lo schema del database e le risorse per la gestione della persistenza dei dati.

## Control
Il package Control contiene le Servlet necessarie all'interazione tra l'utente ed il sistema.

### Control.CartControl
Il package Control.CartControl contiene le servlet responsabili della coordinazione delle operazioni relative alla gestione del carrello, quali l'aggiunta e la rimozione dei prodotti, la visualizzazione del contenuto del carrello e l'aggiornamento delle quantità selezionate.

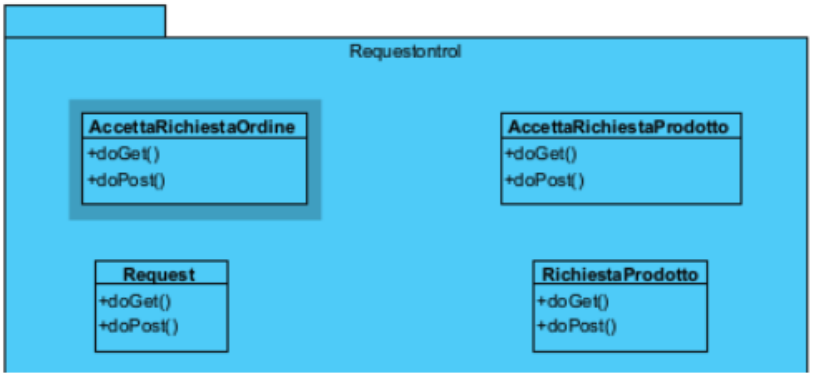



### Control.OrderControl
Il package Control.OrderControl contiene le servlet responsabili della coordinazione delle operazioni relative al completamente dell'acquisto, inclusa la gestione della fase di checkout e la conferma dell'ordine.



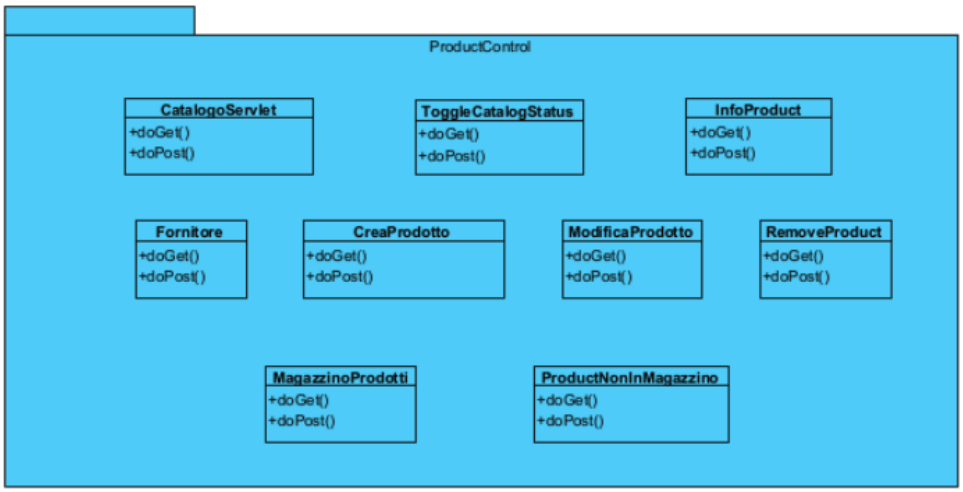
### Control.RequestControl

Il package Control.RequestControl contiene le servlet responsabili della gestione delle richieste verso i fornitori.




### Control.ProductControl

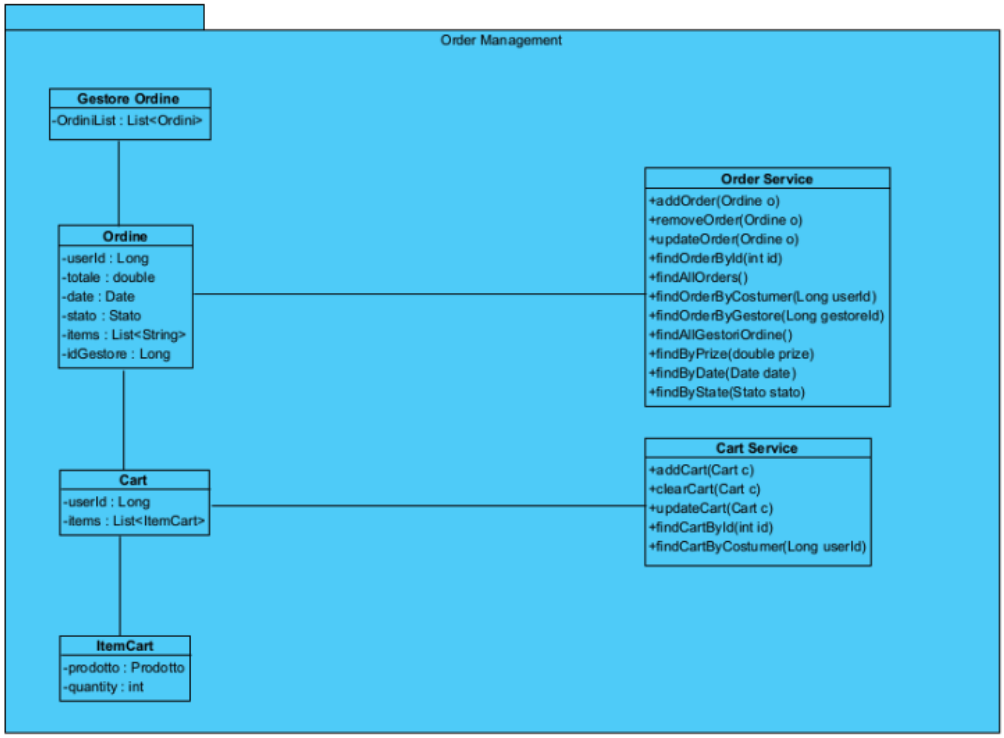
Il package Control.ProductControl contiene le servlet responsabili della gestione delle operazioni relative ai prodotti.



# Packages

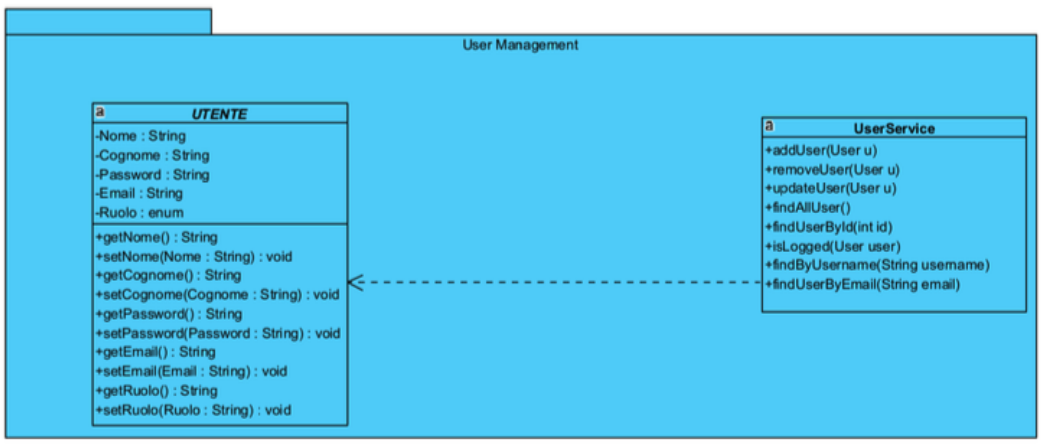
## OrderManagement

Il package OrderManagement contiene le classi Bean e Service dedicate alla gestione e alla finalizzazione degli acquisti effettuati dai clienti.



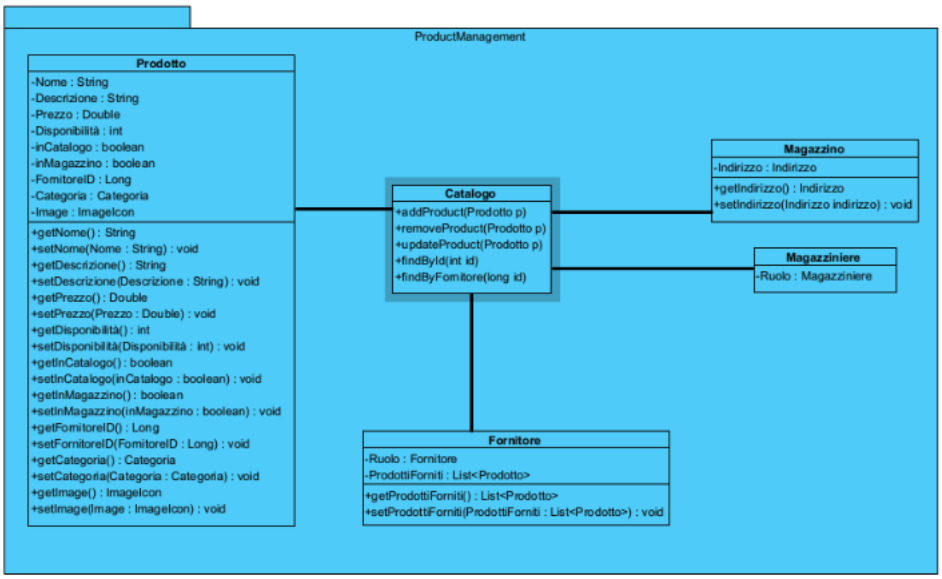

## UserManagement

Il package UserManagement contiene le classi Bean e Service dedicate alla gestione delle informazioni degli utenti e dei relativi account.
L'entità User è definita come classe astratta, all'interno del package sono presente delle implementazioni dei diversi tipi di utente previsti dal sistema.



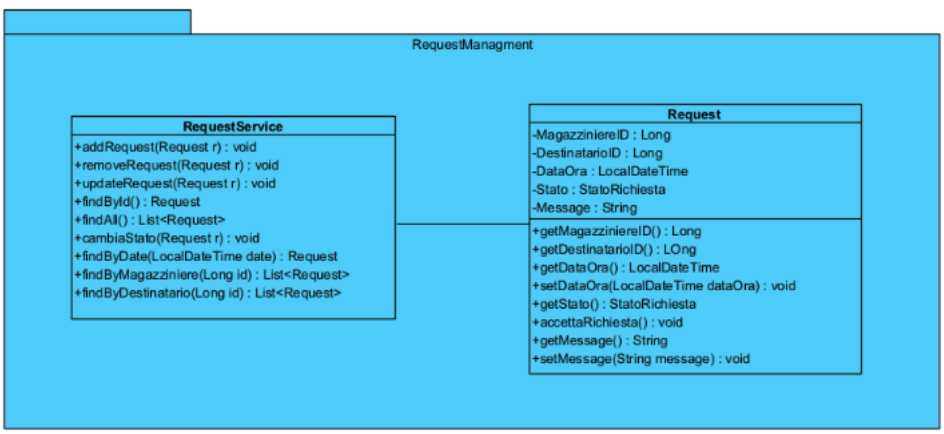
## ProductManagement

Il package ProductManagement contiene le classi Bean e Service dedicate alla gestione delle richieste, sia relative ai prodotti sia alla gestione degli ordini effettuate dai magazzinieri.




## RequestManagement

Il package Request Management contiene le classi Bean e Service dedicate alla gestione delle richieste, sia relative ai prodtti sia alla gestione degli ordini effettuati dai magazzinieri.
L'entità Request è definita come classe astratta, all'interno del package sono presenti le implementazioni delle diverse tipologie di richiesta.



# INTERFACCE DI CLASSE

## CATALOGO

| CATALOGO | |
|---|---|
| **Descrizione** | Questa classe, interfacciandosi con il database, consente di effettuare operazioni di ricerca sui prodotti presenti nel magazzino. |
| **Pre-condizioni** | context Catalogo::findBy(ID:Integer)<br>Pre: ID<>null and ID>0<br><br>context Catalogo::findBy(price:Integer)<br>Pre: price>=0<br><br>context Catalogo::findByName(name:String)<br>Pre: name<>null<br><br>context Catalogo::addProduct(prodotto:Prodotto) pre:<br>prodotto<>null and<br>magazzino::containsProduct(prodotto) and<br>not self.productList -> includes(prodotto)<br><br>context Catalogo::removeProduct(prodotto:Prodotto) pre:<br>prodotto<>null and<br>magazzino::containsProduct(prodotto) and<br>self.productList->includes(prodotto)<br><br>context Catalogo::showProducts()<br>Pre: true<br><br>context Catalogo::updateProduct(prodotto:Prodotto) pre:<br>prodotto<>null and<br>self.productList->include(prodotto)<br><br>context Catalogo::isAvailable(prodotto:Prodotto) pre:<br>prodotto<>null and<br>self.productList->include(prodotto)<br><br>Context Catalogo :: getProductsInCatalogo()<br>Pre: true<br><br>Context Catalogo :: getProductsInMagazzino()<br>Pre: true |
| | **context** Catalogo::findBy(ID:Integer) **post**:<br>if Catalogo.productList->exists( p \| p.id=ID) then<br>result = self.ProdottiList->any(p \| p.id = ID)<br>else<br>result=null<br>endif |

| | |
|---|---|
| **Post-condizioni** | context Catalogo::findBy(price:Integer) post:<br>if Catalogo.productList->exists( p \| p.prezzo<price) then<br>result = self.ProdottiList->all(p \| p.prezzo<price)<br>else<br>result=null<br>endif<br><br>context Catalogo::findByName(name:String) post:<br>if Catalogo.productList->exists( p \| p.nome=name) then<br>result = self.ProdottiList->any(p \| p.nome=name) else<br>result=null<br>endif<br><br>context Catalogo::addProduct(prodotto:Prodotto) post:<br>self.productList=self@pre.productList->including(prodotto)<br><br>context Catalogo::removeProduct(prodotto:Prodotto) post:<br>self.productList=self@pre.productList->excluding(prodotto)<br><br>context Catalogo::showProducts() post:<br>result=self.productList ->forAll(p \| p <> null)<br><br>context Catalogo::updateProduct(prodotto:Prodotto) post:<br>let oldProduct = self.productList ->select(p \| p.ID = prodotto.ID)->first() self.ProdottiList = self.productList @pre->excluding(oldProduct)-> including(prodotto)<br><br>context Catalogo::isAvailable(prodotto:Prodotto) post:<br>result= self.productList->all( prodotto \| prodotto.disponibilità>0 )<br><br>Context Catalogo :: getProductsInCatalogo()<br>Post: result->forall(p \| p.inCatalogo = true)<br><br>Context Catalogo :: getProductsInMagazzino()<br>Post: result->forall(p \| p.inMagazzino = true) |
| **Invarianti** | |

### CartService

| CartService | |
|---|---|
| **Descrizione** | Questa classe, interfacciandosi con il database, gestisce la persistenza del carrello tra diverse sessioni utente. |
| | context CartServiceRemote::addCart(cart: Cart)<br>pre: cart <> null<br><br>context CartServiceRemote::clearCart(cart: Cart)<br>pre: cart <> null and Cart.allInstances()->includes(cart)<br><br>context CartServiceRemote::updateCart(cart: Cart)<br>pre: cart <> null and Cart.allInstances()->includes(cart) |

| | |
|---|---|
| **Pre-condizioni** | context CartServiceRemote::findCartById(id: Integer)<br>pre: id <> null<br><br>context CartServiceRemote::findCartByCostumer(userId: Long)<br>pre: userId > 0 and User.allInstances()->exists(u)<br><br>context CartServiceRemote::removeProductFromCart(cartId: Integer, productId: Integer)<br>pre: cartId > 0 and productId > 0 and Cart.allInstances()->exists(c) |
| **Post-condizioni** | context CartServiceRemote::addCart(cart: Cart)<br>post: Cart.allInstances()->includes(cart)<br><br>context CartServiceRemote::clearCart(cart: Cart)<br>post: cart.products->isEmpty()<br><br>context CartServiceRemote::updateCart(cart: Cart)<br>post: Cart.allInstances()->exists(c)<br><br>context CartServiceRemote::findCartById(id: Integer)<br>post: result = Cart.allInstances()->any(c)<br><br>context CartServiceRemote::findCartByCostumer(userId: Long)<br>post: result = Cart.allInstances()->any(c)<br><br>context CartServiceRemote::removeProductFromCart(cartId: Integer, productId: Integer)<br>post: Cart.allInstances()->exists(c) |
| **Invarianti** | |

### OrderService

| OrderService | |
|---|---|
| **Descrizione** | Questa classe consente ai clienti di accedere e manipolare le informazioni relative agli ordini memorizzati nel database. |
| **Pre-condizioni** | Context OrdineService :: addOrder(o : Ordine)<br>Pre: o<>null<br>And o.userID <> null and Utente.allInstances() -> exists (u\|u.ID=userID)<br>And o.ID<>null and o.ID>0<br>And o.Data<>null<br>And o.Totale<> null and o.Totale>0<br><br>Context OrdineService :: removeOrder(id: Integer)<br>Pre: id<>null and id>0 and Ordine.allInstances()□exists(o\|o.id=id)<br><br>Context OrdineService findOrdersByCustomer (u: Utente)<br>Pre: u<>null<br><br>Context OrdineService findByDate(d: Date)<br>Pre: d<>null |

| | |
|---|---|
| | Context OrdineService updateOrder(o: Ordine)<br>Pre: o<>null and Ordine.allInstances() -> exists(or\| or.id = o.id) |
| **Post-condizioni** | Context OrdineService :: addOrder(o : Ordine)<br>Post: Ordine.allInstances() -> exists(or \| or.ID = o.ID)<br><br>Context OrdineService :: removeOrder(id: Integer)<br>Post: not (Ordine.allIstances() -> exists(o\| o.id=id))<br><br>Context OrdineService findOrdersByCustomer (u: Utente)<br>Post: result= [Ordine.allIIstances() -> any(o\| o.userID = u.ID)]<br><br>Context OrdineService findByDate(d: Date)<br>Post: result= Ordine.allInstances() -> select (o \| o.date= d)<br><br>context OrdineService::updateOrder(o: Ordine): Boolean<br>post:<br>(Ordine.allInstances() -> exists(ord \| ord.id = o.id) implies<br>let updatedOrder = Ordine.allInstances() -> any(ord \| ord.id = o.id) in<br>updatedOrder.data = o.data and<br>updatedOrder.cliente = o.cliente and<br>updatedOrder.totale = o.totale and<br>result = true)<br>and<br>(not Ordine.allInstances()->exists(ord \| ord.id = o.id) implies result = false) |
| **Invariante** | |

### RequestService

| RequestService | |
|---|---|
| **Descrizione** | Questa classe consente ai clienti di accedere e manipolare le informazioni relative alle richieste prodotti e alla gestione degli ordini memorizzate nel database. |
| **Pre-condizioni** | Context RequestService :: addRequest(r : Request)<br>Pre: r <> null<br><br>and r.magazziniereID <> null<br>and r.destinatarioID <> null<br>and (r.statoRichiesta = 'accettato' or r.statoRichiesta = 'non accettato')<br><br>Context RequestService:: removeRequest(r: Request)<br>Pre: r<>null<br><br>Context RequestService :: updateRequest(r: RequestService)<br>Pre: r<>null<br><br>Context RequestService :: findById(id: Long)<br>Pre: id <> null and Request.allInstances()->exists(r \| r.id = id)<br><br>Context RequestService :: cambiaStato(r: Request)<br>Pre: id <> null and Request.allInstances()->exists(r \| r.id = id) |

| | |
|---|---|
| | Context RequestService :: findByMagazziniere (idMagazziniere: Long)<br>Pre: id <> null and<br>Magazziniere.allInstances()->exists(m \| m.id = idMagazziniere)<br><br>Context RequestService :: findByDestinatario(idDestinatario: Long)<br>Pre: idDestinatario <> null and<br>(Fornitore.allInstances()->exists(f \| f.id = idDestinatario) or<br>GestoreOrdini.allInstances()->exists(g \| g.id = idDestinatario)) |
| **Post-condizioni** | Context RequestService :: addRequest(r : Request) Post:<br>Request.allInstances() ->includes(r)<br><br>Context RequestService :: removeRequest(r : Request) Post: not<br>Request.allInstances() -> includes(r)<br><br>Context RequestService :: updateRequest(r: RequestService)<br>Post: Request.allInstances() -> exists(req \| req.id = r.id and req = r)<br><br>Context RequestService :: findById(id: Long)<br>Post: result <> null implies Request.allInstances()->exists(r \| r.id = id and r = result)<br><br>Context RequestService :: cambiaStato(r: Request)<br>Post: r.stato <> r.stato@pre<br><br>Context RequestService :: findByMagazziniere (idMagazziniere: Long)<br>Post: result -> forall(r \| r.magazziniereID = idMagazziniere)<br><br>Context RequestService :: findByDestinatario(idDestinatario: Long)<br>Post: result -> forall(r \| r.destinatarioID = idDestinatario) |
| **Invarianti** | |