



iSnapGaming

ODD

<b>ODD</b>	<b>3</b>
<b>Introduction</b>	<b>3</b>
<b>Object design trade-offs</b>	<b>3</b>
1.1 Robustezza vs Velocità di implementazione	3
1.2 Chiarezza vs Velocità di implementazione	3
Interface documentation guidelines	3
References	4
<b>Directories</b>	<b>4</b>
<b>Design Patterns</b>	<b>7</b>
<b>Packages</b>	<b>8</b>
<b>Class Interfaces</b>	<b>9</b>
User Management	10
User	10
Customer	10
Manager	10
OrderManager	11
ProductManager	12
Address	14
Product Management	15
Product	15
Order Management	16
Cart	16
CustomerOrder	16
ItemCart	18
OrderProduct	18
OrderCreation	18
Storage Management	19
UserDAO	19
CustomerDAO	21
ProductDAO	22
AddressDAO	23
Description	23
doSave	23
findByKey	23
findByCustomerId	23
CustomerOrderDAO	24
PaymentManagement	25
PaymentAuthorizationServiceAdapter	25
<b>Object Design Model Optimization</b>	<b>26</b>

# ODD

## Introduction

L'Object Design Document (ODD) è un documento essenziale nel processo di sviluppo software. Tale documento si basa sui documenti RAD e SDD, che consolidano le informazioni raccolte durante l'analisi dei requisiti e la progettazione, l'ODD offre una guida dettagliata su come strutturare e implementare il sistema. Descrive le interfacce delle classi, le operazioni supportate, i tipi di dati utilizzati, i parametri delle procedure, i signature dei sottosistemi, trade-off . Questo documento include anche strategie per gestire compromessi di progettazione durante lo sviluppo, fungendo da "piano di costruzione" per il software.

## Object design trade-offs

### 1.1 Robustezza vs Velocità di implementazione

Nel gestire i dati in ingresso, sappiamo quanto sia importante controllarli bene. Tuttavia, per rilasciare la prima versione del sistema il più velocemente possibile, abbiamo deciso di accettare che i controlli iniziali non siano perfetti. Ci impegniamo comunque a migliorare questi controlli nelle versioni future del sistema. Questa scelta ci permette di uscire rapidamente con la prima versione, sapendo che rafforzeremo la robustezza nei prossimi aggiornamenti.

### 1.2 Chiarezza vs Velocità di implementazione

Per semplificare il testing, sarebbe ideale scrivere codice molto chiaro. Tuttavia, con i tempi di sviluppo stretti per questa prima versione, non sempre potremo mantenere i nostri standard abituali. Nelle versioni future del sistema, potremo migliorare questo aspetto.

## Interface documentation guidelines

1. Le classi hanno dei nomi comuni singolari.
2. I metodi sono denominati con frasi verbali.
3. Gli Error Status sono restituiti attraverso eccezioni.

## References

- SDD: ci si riferisce al SDD quando si spiega l'organizzazione dei package, dato che quest' ultima è stata creata proprio a partire dalla suddivisione in subsystem.

## Directories

main > java > com.isnappgaming > OrderManagement

- Cart.java
- CustomerOrder.java
- ItemCart.java
- OrderProduct.java
- OrderCreation.java

main > java > com.isnappgaming > PaymentManagement

- PaymentAuthorizationService.java
- PaymentAuthorizationServiceAdapter.java
- PaymentAuthServiceInterface.java

main > java > com.isnappgaming > ProductManagement

- Product.java

main > java > com.isnappgaming > StoreManagement.DAO

- AddressDAO.java
- CustomerDAO.java
- CustomerOrderDAO.java
- ProductDAO.java
- UserDAO.java

main > java > com.isnappgaming > UserManagement

- Address.java
- Customer.java
- Manager.java
- OrderManager.java
- ProductManager.java
- User.java

main > java > com.isnapgaming > utils

- AccessControlFilter.java
- ContextInitializer.java
- ImageServlet.java
- TestPopolamento.java

main > java > com.isnapgaming > view > Customer

- PayOrder.java

main > java > com.isnapgaming > view > Guest

- AddToCart.java
- ProductDetails.java
- Signup.java
- UpdateCart.java

main > java > com.isnapgaming > view > Manager

- RoleSelection.java

main > java > com.isnapgaming > view > OrderManager

- GetOrderDetails.java
- GetOrdersList.java
- UpdateStatus.java

main > java > com.isnapgaming > view > ProductManager

- AddProduct.java
- ManageProduct.java
- ProductAvailability.java
- ProductUpdater.java

main > java > com.isnapgaming > view > User

- Login.java
- Logout.java

main > java > testing

Sotto tale directory sono presenti tutti i file per poter inizializzare il database prima di ogni test case durante le attività di testing:

- RetrieveCredentials.java
- SQLScript.java

main > webapp > app\_imgs > logo

- logo.png

main > webapp > app\_imgs > products

Sotto tale directory sono presenti tutti i file riguardanti i prodotti:

- 111\_1.jpg
- 123\_1.jpg
- 157\_1.jpg
- 268\_1.jpg
- 331\_1.jpg
- 395\_1.jpg
- 482\_1.jpg
- 556\_1.jpg
- 652\_1.jpg
- 751\_1.jpg
- 812\_1.jpg
- 987\_1.jpg

main > webapp > fragments

- header.jsp
- footer.jsp

main > webapp > META-INF

- context.xml

main > webapp > scripts

- validate.js

main > webapp > styles

- access.css
- addProduct.css
- cart.css

- checkout.css
- confirmationPage.css
- errorPage.css
- footer.css
- header.css
- index.css
- orderDetails.css
- orderManagerDashboard.css
- product.css
- productManagerDashboard.css
- roleSelection.css
- updateStatus.css

main > webapp > WEB-INF

- web.xml

main > webapp

- addProduct.jsp
- cart.jsp
- checkout.jsp
- confirmationPage.jsp
- customerOrdersList.jsp
- errorPage.jsp
- index.jsp
- login.jsp
- orderDetails.jsp
- orderManagerDashboard.jsp
- product.jsp
- productDetailsPM.jsp
- productManagerDashboard.jsp
- roleSelection.jsp
- signup.jsp
- updateProduct.jsp
- updateStatus.jsp

## Design Patterns

All'interno del nostro progetto sono stati utilizzati dei design pattern per riuscire a implementare alcune funzionalità del sistema.

Ad esempio abbiamo ritenuto utile andare a utilizzare un pattern di tipo DAO che prevede la creazione di una classe DAO per ogni tipo di oggetto per il quale si desidera gestire la persistenza. Ogni classe DAO si occupa di gestire l'accesso ai dati per quel tipo di oggetto e fornisce metodi per eseguire le operazioni come il recupero, l'inserimento, l'aggiornamento e

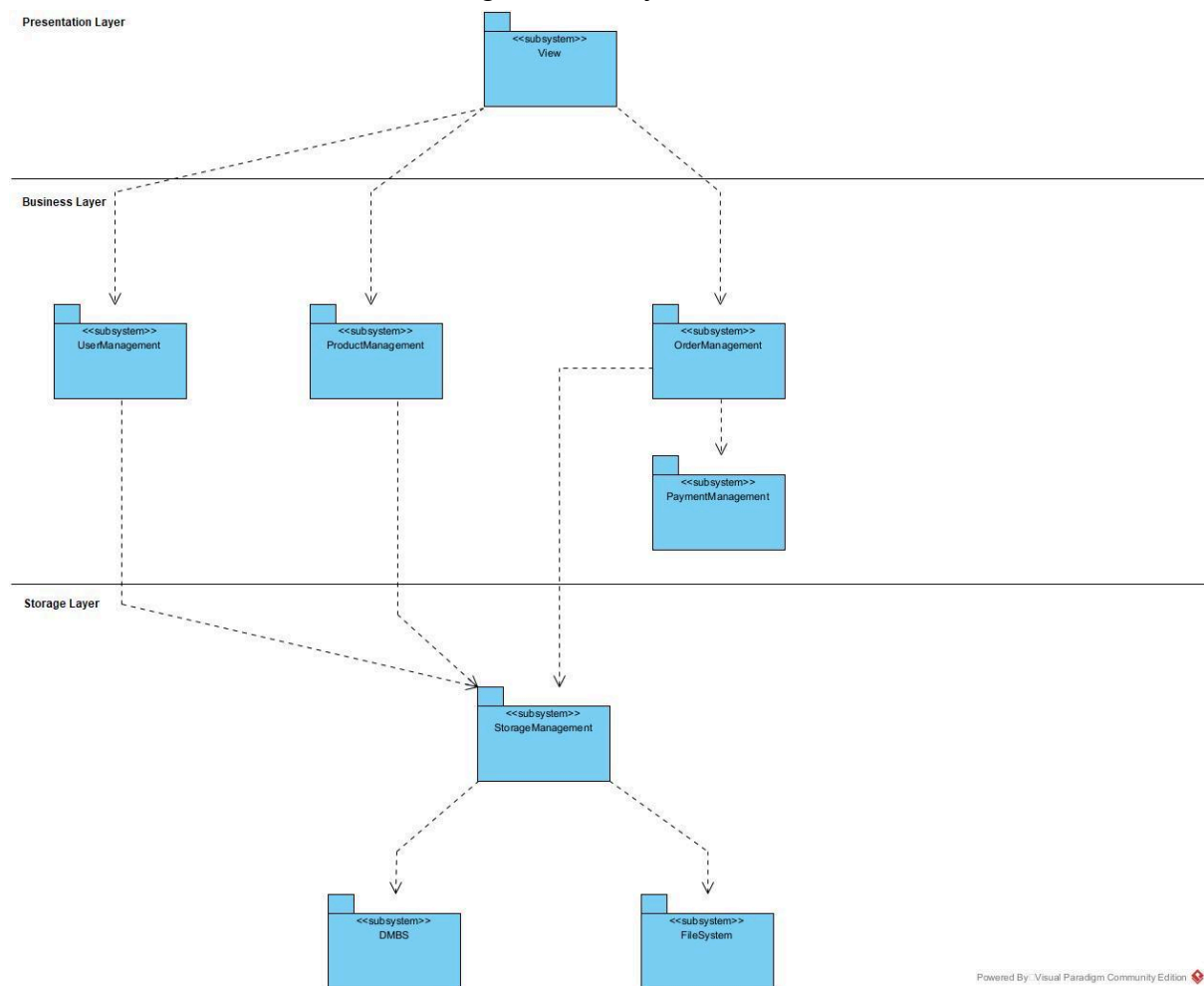
la cancellazione dei dati. L'obiettivo principale del pattern DAO è quello di separare l'accesso ai dati dall'applicazione stessa, in modo da poter modificare o sostituire facilmente la fonte di dati senza dover modificare il codice dell'applicazione.

Abbiamo anche pensato che fosse utile adottare un pattern di tipo interceptor per implementare il filtro che controlla i permessi necessari per eseguire una certa funzionalità. In tal modo, intercetteremo la richiesta e la rifiuteremo se il permesso richiesto non è presente.

Infine, abbiamo optato per un pattern adapter per incapsulare la logica di pagamento. Questa logica viene gestita tramite servizi forniti da terze parti, non sviluppati direttamente da noi. Utilizzare questo pattern è essenziale per evitare la dipendenza da quei servizi e dalle loro implementazioni, consentendoci di mantenere un disaccoppiamento efficace.

## Packages

Il sistema è stato suddiviso nei seguenti subsystems:



A partire da questo subsystem sono stati quindi prodotti i seguenti packages:

1. ProductManagement



2. OrderManagement
3. PaymentManagement
4. UserManagement
5. StorageManagement
6. view

1. All'interno di ProductManagement sono presenti le classi per la gestione del prodotto. Quest'ultimo è mappabile direttamente al sottosistema ProductManagement.

2. All'interno di OrderManagement sono presenti le classi per la gestione dell'ordine. Quest'ultimo è mappabile direttamente al sottosistema OrderManagement.

3. All'interno di PaymentManagement sono presenti le classi per la gestione del pagamento. Quest'ultimo è mappabile direttamente al sottosistema PaymentManagement.

4. All'interno di UserManagement sono presenti le classi per la gestione dell'utente. Quest'ultimo è mappabile direttamente al sottosistema UserManagement.

5. All'interno di StorageManagement sono presenti tutte le classi necessarie per riuscire a garantire la persistenza dei dati. Quest'ultimo è direttamente mappabile al sottosistema StorageManagement. Il seguente package possiede delle dipendenze con i package UserManagement, OrderManagement, ProductManagement.

6. All'interno di view si trovano tutti i package che contengono le servlet, le quali gestiscono la logica di controllo dell'applicazione. Le servlet sono organizzate nei vari package sulla base del ruolo di cui ci permettono di realizzare le funzionalità. Questo package, ha una dipendenza con tutti gli altri perché realizzando la logica di controllo invoca i metodi delle classi che rappresentano i dati e permettono di effettuare la persistenza dei dati.

## Class Interfaces

Nella specifica delle interfacce abbiamo deciso di non andare a specificare ogni getter e setter presenti nelle classi entity e, inoltre, di non andare a effettuare la specifica delle interfacce per tutti i metodi privati delle classi.

Non abbiamo effettuato la specifica delle interfacce per tutti i metodi privati delle classi.

Le classi DAO sono le uniche classi che accedono al database.

## User Management

User	
Description	La classe User rappresenta un utente con ruolo generico registrato all'interno del sistema.
Invariant	<b>context</b> User <b>inv:</b> username <> null & password <> null & firstName <> null & lastName <> null & dateOfBirth <> null
makeUser	<b>context</b> User::makeUser(username: String, password: String, firstName: String, lastName: String, dateOfBirth: LocalDate) <b>pre:</b> username <> null & password <> null & firstName <> null & lastName <> null & dateOfBirth <> null  <b>context</b> User::makeUser(username: String, password: String, firstName: String, lastName: String, dateOfBirth: LocalDate) <b>post:</b> result.username = username & result.password = password & result.firstName = firstName & result.lastName = lastName

Customer	
Description	Rappresenta un utente registrato con ruolo Customer.
Invariant	<b>context</b> Customer <b>inv:</b> -

Manager	
Description	Rappresenta un manager generico all'interno del sistema
Invariant	<b>context</b> Manager <b>inv:</b> -

OrderManager	
Description	Classe che rappresenta un OrderManager.
Invariant	<b>context</b> OrderManager <b>inv:</b> -
getAllCustomerOrders	<p>Metodo che restituisce tutti gli ordini presenti all'interno del database.</p> <p><b>context</b> OrderManager::getAllCustomerOrders(dataSource: DataSource) <b>pre:</b> -</p> <p><b>context</b> OrderManager::getAllCustomerOrders(dataSource: DataSource) <b>post:</b> -</p>
checkProduct	<p>Metodo che permette di aggiornare lo stato dell'ordine in <i>UnderPreparation</i></p> <p><b>context</b> OrderManager::checkProduct(order: CustomerOrder, dataSource: DataSource) <b>pre:</b> order &lt;&gt; null &amp; dataSource &lt;&gt; null, &amp; order.getStatus() = "TO_BE_MANAGED"</p> <p><b>context</b> OrderManager::checkProduct(order: CustomerOrder, dataSource: DataSource) <b>post:</b> order.getStatus() = "UNDER_PREPARATION"</p>
packProduct	<p>Metodo che permette di aggiornare lo stato dell'ordine in <i>ReadyForSending</i>.</p> <p><b>context</b> OrderManager::packProduct(order: CustomerOrder, dataSource: DataSource) <b>pre:</b> order &lt;&gt; null &amp; dataSource &lt;&gt; null &amp; order.getStatus() = "UNDER_PREPARATION"</p> <p><b>context</b> OrderManager::packProduct(order: CustomerOrder, dataSource: DataSource) <b>post:</b> order.getStatus() = "READY_FOR_SENDING"</p>
replaceProduct	<p>Metodo che permette di aggiornare lo stato dell'ordine in <i>UnderPreparation</i>.</p> <p><b>context</b> OrderManager::replaceProduct(order: CustomerOrder, dataSource: DataSource) <b>pre:</b> &amp; order &lt;&gt; null &amp; dataSource &lt;&gt; null</p>

	<p>order.getStatus() = "READY_FOR_SENDING"</p> <p><b>context</b> OrderManager::replaceProduct(order: CustomerOrder, dataSource: DataSource) <b>post:</b> order.getStatus() = "UNDER_PREPARATION"</p>
<b>contactCourier</b>	<p>Metodo che permette di aggiornare lo stato dell'ordine in <i>Shipped</i>.</p> <p><b>context</b> OrderManager::contactCourier(order: CustomerOrder, dataSource: DataSource) <b>pre:</b> order &lt;&gt; null &amp; dataSource &lt;&gt; null &amp; order.getStatus() = "READY_FOR_SENDING"</p> <p><b>context</b> OrderManager::contactCourier(order: CustomerOrder, dataSource: DataSource) <b>post:</b> order.getStatus() = "SHIPPED"</p>
<b>restoreOrder</b>	<p>Metodo che permette di aggiornare lo stato dell'ordine in <i>UnderPreparation</i>.</p> <p><b>context</b> OrderManager::restoreOrder(order: CustomerOrder, dataSource: DataSource) <b>pre:</b> order &lt;&gt; null &amp; dataSource &lt;&gt; null &amp; order.getStatus() = "SHIPPED"</p> <p><b>context</b> OrderManager::restoreOrder(order: CustomerOrder, dataSource: DataSource) <b>post:</b> order.getStatus() = "UNDER_PREPARATION"</p>
<b>confirmDelivery</b>	<p><b>context</b> OrderManager::confirmDelivery(order: CustomerOrder, dataSource: DataSource) <b>pre:</b> order &lt;&gt; null &amp; dataSource &lt;&gt; null &amp; order.getStatus() = "SHIPPED"</p> <p><b>context</b> OrderManager::confirmDelivery(order: CustomerOrder, dataSource: DataSource) <b>post:</b> order.getStatus() = "DELIVERED"</p>

ProductManager	
<b>Description</b>	Rappresenta un Product Manager che gestisce i prodotti all'interno del sistema
<b>Invariant</b>	<b>context</b> ProductManager <b>inv:</b> -
<b>addProduct</b>	Metodo che permette di inserire un nuovo prodotto nel

	<p>sistema.</p> <p><b>context</b> ProductManager::addProduct(product: Product, dataSource: DataSource) <b>pre:</b>  product &lt;&gt; null  &amp; dataSource &lt;&gt; null</p> <p><b>context</b> ProductManager::addProduct(product: Product, dataSource: DataSource) <b>post:</b></p> <p>this.getProductByProdCode(product.getProdCode(), DataSource) = product</p>
<b>updateProduct</b>	<p>Metodo che permette di aggiornare un prodotto già presente nel sistema.</p> <p><b>context</b> ProductManager::updateProduct(product: Product, dataSource: DataSource) <b>pre:</b>  product &lt;&gt; null  &amp; dataSource &lt;&gt; null</p> <p><b>context</b> ProductManager::updateProduct(product: Product, dataSource: DataSource) <b>post:</b></p> <p>this.getProductByProdCode(product.getProdCode(), DataSource) = product</p>
<b>removeProduct</b>	<p>Metodo che permette di rimuovere un prodotto dal catalogo rendendolo, cioè, non disponibile alla vendita.</p> <p><b>context</b> ProductManager::removeProduct(product: Product, dataSource: DataSource) <b>pre:</b>  product &lt;&gt; null  &amp; dataSource &lt;&gt; null</p> <p><b>context</b> ProductManager::removeProduct(product: Product, dataSource: DataSource) <b>post:</b></p> <p>this.getProductByProdCode(product.getProdCode(), DataSource).isAvailable = false</p>
<b>makeProductAvailable</b>	<p>Metodo che permette di rendere un prodotto disponibile alla vendita.</p> <p><b>context</b> ProductManager::makeProductAvailable(product: Product, dataSource: DataSource) <b>pre:</b>  product &lt;&gt; null  &amp; dataSource &lt;&gt; null</p> <p><b>context</b> ProductManager::makeProductAvailable(product:</p>

	Product, dataSource: DataSource) <b>post:</b>  this.getProductByProdCode(product.getProdCode(), DataSource).isAvailable = true
<b>getProductByProdCode</b>	Metodo che restituisce il prodotto che ha come prodCode quello specificato.  <b>context</b> ProductManager::getProductByProdCode(prodCode: Integer, dataSource: DataSource) <b>pre:</b> prodCode > 0 & dataSource <> null  <b>context</b> ProductManager::getProductByProdCode(prodCode: Integer, dataSource: DataSource) <b>post:</b> result <> null & result.prodCode = prodCode
<b>getAllProducts</b>	Metodo che restituisce tutti i prodotti presenti nel sistema.  <b>context</b> ProductManager::getAllProducts(dataSource: DataSource) <b>pre:</b> dataSource <> null  <b>context</b> ProductManager::getAllProducts(dataSource: DataSource) <b>post:</b> -

Address	
<b>Description</b>	Rappresenta un indirizzo associato ad un Customer.
<b>Invariant</b>	<b>context</b> Address <b>inv:</b> customerId > 0 & street <> null & city <> null & postalCode > 0
<b>makeAddress</b>	<b>context</b> Address::makeAddress(customerId: Integer, street: String, city: String, postalCode: Integer) <b>pre:</b> customerId > 0 & street <> null & city <> null & postalCode > 0  <b>context</b> Address::makeAddress(customerId: Integer, street: String, city: String, postalCode: Integer) <b>post:</b> result.customerId = customerId & result.street = street

	& result.city = city & result.postalCode = postalCode
--	--

## Product Management

Product	
Description	Rappresenta un generico prodotto inserito nel sistema.
Invariant	<b>context</b> Product inv: -
makeProduct	<b>context</b> Product::makeProduct(prodCode: Integer, name: String, softwareHouse: String, platform: Product.Platform, price: Integer, quantity: Integer, category: Product.Category, pegi: Product.Pegi, releaseYear: Integer, imagePath: String) <b>pre:</b> prodCode > 0 & name <> null & softwareHouse <> null & platform <> null & price >= 0 & quantity >= 0 & category <> null & pegi <> null & releaseYear <> null & imagePath <> null  <b>context</b> Product::makeProduct(prodCode: Integer, name: String, softwareHouse: String, platform: Product.Platform, price: Integer, quantity: Integer, category: Product.Category, pegi: Product.Pegi, releaseYear: Integer, imagePath: String) <b>post:</b> result.prodCode = prodCode & result.name = name & result.softwareHouse = softwareHouse & result.platform = platform & result.price = price & result.quantity = quantity & result.category = category & result.pegi = pegi & resut.releaseYear = releaseYear & result.imagePath = imagePath

## Order Management

Cart	
Description	Rappresenta il carrello associato ad un Customer.
Invariant	<b>context</b> Cart <b>inv</b> : items <> null & totalPrice >= 0
addToCart	<p>Metodo che consente di aggiungere un prodotto al carrello. Se il prodotto è già presente, allora ne viene semplicemente aggiornata la quantità.</p> <p><b>context</b> Cart::addToCart(product: Product, quantity: Integer) <b>pre</b>: product &lt;&gt; null &amp; quantity &gt; 0</p> <p><b>context</b> Cart::addToCart(product: Product, quantity: Integer) <b>post</b>: this.getItems().includes(product) = True</p>
removeFromCart	<p>Metodo che consente di rimuovere un prodotto dal carrello.</p> <p><b>context</b> Cart::removeFromCart(product: Product) <b>pre</b>: product &lt;&gt; null &amp; this.getItems().includes(product) = True</p> <p><b>context</b> Cart::removeFromCart(product: Product) <b>post</b>: this.getItems().includes(product) = False</p>
decreaseQuantityCart	<p>Metodo che consente di decrementare la quantità di un prodotto nel carrello.</p> <p><b>context</b> Cart:decreaseQuantityCart(product: Product, quantity: Integer) <b>pre</b>: product &lt;&gt; null &amp; this.getItems().includes(product) = True</p> <p><b>context</b> Cart:decreaseQuantityCart(product: Product, quantity: Integer) <b>post</b>: this.getItems().includes(product) = false    quantità del carrello decrementata di quantity</p>

CustomerOrder	
Description	Rappresenta un ordine associato ad un Customer.



<b>Invariant</b>	<b>context</b> CustomerOrder <b>inv:</b> id > 0 & customerId <> null & (status = "TO_BE_MANAGED"    status = "UNDER_PREPARATION"    status = "READY_FOR_SENDING"    status = "SHIPPED",    status = "DELIVERED") & address <> null & orderDate <> null & totalAmount > 0 & products <> null
<b>makeCustomerOrder</b>	Metodo che consente di creare un oggetto CustomerOrder con i campi indicati.  <b>context</b> CustomerOrder::makeCustomerOrder(customerId: Integer, address: String, orderDate: LocalDate, products: List<OrderProduct>) <b>pre:</b> customerId <> null & customerId <> 0 & address <> null & address <> "" & orderDate <> null & products<> null  <b>context</b> CustomerOrder::makeCustomerOrder(customerId: Integer, address: String, orderDate: LocalDate, products: List<OrderProduct>) <b>post:</b> this.getCustomerId() = customerId & this.getStatus() = "TO_BE_MANAGED" & this.getAddress() = address & this.getOrderDate() = orderDate, & this.getProducts() = products
<b>addProduct</b>	Metodo che consente di aggiungere un prodotto alla lista dei prodotti associati all'ordine.  <b>context</b> CustomerOrder::addProduct(product: Product) <b>pre:</b> product <> null  <b>context</b> CustomerOrder::addProduct(product: Product) <b>post:</b> this.getProducts().includes(product) = True
<b>calculateTotalAmount</b>	Metodo che calcola il prezzo totale da pagare.  <b>context</b> CustomerOrder::calculateTotalAmount() <b>pre:</b> -  <b>context</b> CustomerOrder::calculateTotalAmount() <b>post:</b> totalAmount > 0

ItemCart	
<b>Description</b>	Classe che rappresenta un particolare prodotto nel carrello di un customer.
<b>Invariant</b>	<b>context</b> ItemCart <b>inv</b> : cart<> null & product <> null & quantity > 0
<b>getTotalPrice</b>	Metodo che calcola il costo totale di un prodotto in relazione alla quantità inserita.  <b>context</b> ItemCart::getTotalPrice() <b>pre</b> : - <b>context</b> ItemCart::getTotalPrice() <b>post</b> : getTotalPrice() = this.price * this.quantity

OrderProduct	
<b>Description</b>	Classe che rappresenta un particolare prodotto presente in un CustomerOrder.
<b>Invariant</b>	<b>context</b> OrderProduct <b>inv</b> : orderId > 0 & productId > 0 & quantity > 0
<b>getTotalPrice</b>	Metodo che calcola il costo totale di un prodotto in relazione alla quantità presente nell'ordine.  <b>context</b> OrderProduct::getTotalPrice() <b>pre</b> : - <b>context</b> OrderProduct::getTotalPrice() <b>post</b> : getTotalPrice() = this.price * this.quantity

OrderCreation	
<b>Description</b>	Classe che contiene la logica di business relativa alla creazione di un ordine e al pagamento di quest'ultimo, utilizzata nelle fasi finali del processo di acquisto di un prodotto.
<b>Invariant</b>	<b>context</b> OrderCreation <b>inv</b> : cart <> null & customer <> null & address <> null
<b>makeOrder</b>	Metodo che crea un ordine al momento del checkout.

	<p><b>context</b> makeOrder::makeOrder(cart: Cart, customer: User, address: String) <b>pre:</b>          cart &lt;&gt; null          &amp; customer &lt;&gt; null          &amp; address &lt;&gt; null</p> <p><b>context</b> OrderProduct::getTotalPrice() <b>post:</b>          result.customer&lt;&gt;null          &amp;result.address&lt;&gt;null          &amp;result.orderDate&lt;&gt;null          &amp;result.products&lt;&gt;null</p>
<b>pay</b>	<p>Metodo che permette di finalizzare il pagamento dell'ordine.</p> <p><b>context</b> OrderCreation::pay(order: CustomerOrder, address: String, cardNumber: String, expirationDate: LocalDate, cvv: String) <b>pre:</b>          -</p> <p><b>context</b> OrderCreation::pay(order: CustomerOrder, address: String, cardNumber: String, expirationDate: LocalDate, cvv: String) <b>post:</b>          -</p>

## Storage Management

UserDAO	
<b>Description</b>	Classe DAO relativa alla classe User che consente di gestirne la persistenza.
<b>doSave</b>	<p>Metodo che permette di rendere persistente un User e ne restituisce il relativo id assegnato nel database.</p> <p><b>context</b> UserDAO::doSave(user: User) <b>pre:</b>          user &lt;&gt; null</p> <p><b>context</b> UserDAO::doSave(user: User) <b>post:</b>          this.findByKey(user.getId()) = user</p>
<b>getUserByUsernameAndPassword</b>	<p>Metodo che restituisce, se esiste, l'utente che ha username e password specificati.</p> <p><b>context</b>          UserDAO::getUserByUsernameAndPassword(username: String, password: String) <b>pre:</b>          username &lt;&gt; null          &amp; username &lt;&gt; ""</p> <p><b>context</b></p>

	<p>UserDAO::getUserByUsernameAndPassword(username: String, password: String) <b>post:</b> Se username e password corrispondono a un utente presente nel database allora:              result = user          altrimenti              result = null</p>
<b>getUserRoles</b>	<p>Metodo che restituisce i ruoli associati ad un utente.</p> <p><b>context</b> UserDAO::getUserRoles(userId: Integer) <b>pre:</b>              userId &gt; 0</p> <p><b>context</b> UserDAO::getUserRoles(userId: Integer)  <b>post:</b>              result &lt;&gt; null</p>
<b>isCustomer</b>	<p>Metodo che restituisce true se un utente è un Customer, false altrimenti.</p> <p><b>context</b> UserDAO::isCustomer(userId: Integer) <b>pre:</b>              userId &gt; 0</p> <p><b>context</b> UserDAO::isCustomer(userId: Integer) <b>post:</b>              -</p>
<b>isOrderManager</b>	<p>Metodo che restituisce true se un utente è un OrderManager, false altrimenti.</p> <p><b>context</b> UserDAO::isOrderManager(userId: Integer)  <b>pre:</b>              userId &gt; 0</p> <p><b>context</b> UserDAO::isOrderManager(userId: Integer)  <b>post:</b>              -</p>
<b>isProductManager</b>	<p>Metodo che restituisce true se un utente è un ProductManager, false altrimenti.</p> <p><b>context</b> UserDAO::isOrderManager(userId: Integer)  <b>pre:</b>              userId &gt; 0</p> <p><b>context</b> UserDAO::isOrderManager(userId: Integer)  <b>post:</b>              -</p>
<b>findByKey</b>	<p>Metodo che restituisce l'utente che ha come id quello specificato.</p> <p><b>context</b> UserDAO::findByKey(userId: Integer) <b>pre:</b>              userId &gt; 0</p>

	<b>context</b> UserDao::findByKey(userId: Integer) <b>post:</b> result <> null
<b>findByUsername</b>	<p>Metodo che restituisce l'utente che ha come username quello specificato.</p> <p><b>context</b> UserDao::findByUsername(username: String) <b>pre:</b> username &lt;&gt; 0</p> <p><b>context</b> UserDao::findByUsername(userId: Integer) <b>post:</b></p>

CustomerDAO	
<b>Description</b>	Classe DAO relativa alla classe Customer che consente di gestirne la persistenza.
<b>doSave</b>	<p>Metodo che permette di rendere persistente un Customer e ne restituisce il relativo id assegnato nel database.</p> <p><b>context</b> CustomerDAO::doSave(customer: Customer) <b>pre:</b> customer &lt;&gt; null</p> <p><b>context</b> CustomerDAO::doSave(customer: Customer) <b>post:</b> this.findByKey(customer.getId()) = customer</p>
<b>findByKey</b>	<p>Metodo che restituisce il Customer che ha come id quello specificato.</p> <p><b>context</b> CustomerDAO::findByKey(customerId: Integer) <b>pre:</b> customerId &gt; 0</p> <p><b>context</b> CustomerDAO::doSave(customerId: Integer) <b>post:</b> result &lt;&gt; null</p>
<b>findAddressesByCustomerId</b>	<p>Metodo che restituisce gli addresses associati al customer che ha come id quello specificato.</p> <p><b>context</b> CustomerDAO::findAddressesByCustomerId(customerId: Integer) <b>pre:</b> customerId &gt; 0</p> <p><b>context</b> CustomerDAO::findAddressesByCustomerId(customerId: Integer) <b>post:</b></p>

	result <> null
--	----------------

ProductDAO	
<b>Description</b>	Classe DAO relativa alla classe Product che consente di gestirne la persistenza.
<b>doSave</b>	<p>Metodo che permette di rendere persistente un Product e ne restituisce il relativo id assegnato nel database.</p> <p><b>context</b> ProductDAO::doSave(product: Product) <b>pre:</b> product &lt;&gt; null</p> <p><b>context</b> ProductDAO::doSave(product: Product) <b>post:</b> this.findByKey(product.getId()) = product</p>
<b>doUpdate</b>	<p>Metodo che permette di aggiornare un prodotto nel database.</p> <p><b>context</b> ProductDAO::doUpdate(product: Product) <b>pre:</b> product &lt;&gt; null</p> <p><b>context</b> ProductDAO::doUpdate(product: Product) <b>post:</b> this.findByKey(product.getId()) = product</p>
<b>findByKey</b>	<p>Metodo che restituisce il Prodotto che ha come id quello specificato.</p> <p><b>context</b> ProductDAO::findByKey(productId: Integer) <b>pre:</b> productId &gt; 0</p> <p><b>context</b> ProductDAO::findByKey(productId: Integer) <b>post:</b> result &lt;&gt; null</p>
<b>findByProdCode</b>	<p>Metodo che restituisce il prodotto che ha prodCode uguale a quello specificato.</p> <p><b>context</b> ProductDAO::findByProdCode(productId: Integer) <b>pre:</b> productId &lt;&gt; null</p> <p><b>context</b> ProductDAO::findByProdCode(productId: Integer) <b>post:</b> result &lt;&gt; null</p>

<b>findByCategory</b>	<p>Metodo che restituisce tutti i prodotti appartenenti alla categoria specificata.</p> <p><b>context</b> ProductDAO::findByCategory(category: Product.Category) <b>pre:</b> category &lt;&gt; null</p> <p><b>context</b> ProductDAO::findByCategory(category: Product.Category) <b>post:</b> result &lt;&gt; null</p>
<b>doRetrieveAll</b>	<p>Metodo che restituisce tutti i prodotti presenti nel database.</p> <p><b>context context</b> ProductDAO::doRetriveAll() <b>pre:</b> -</p> <p><b>context context</b> ProductDAO::doRetriveAll()<b>post:</b> -</p>

AddressDAO	
<b>Description</b>	Classe DAO relativa alla classe Address che consente di gestirne la persistenza.
<b>doSave</b>	<p>Metodo che permette di rendere persistente un Address e ne restituisce il relativo id assegnato nel database.</p> <p><b>context</b> AddressDAO::doSave(address: Address) <b>pre:</b> address &lt;&gt; null</p> <p><b>context</b> AddressDAO::doSave(address: Address) <b>post:</b> this.findByKey(address..getId()) = address</p>
<b>findByKey</b>	<p>Metodo che restituisce l'Address che ha come id quello specificato.</p> <p><b>context</b> AddressDAO::findByKey(addressId: Integer) <b>pre:</b> addressId &gt; 0</p> <p><b>context</b> AddressDAO::findByKey(addressId: Integer) <b>post:</b> result.getId() = addressId</p>
<b>findByCustomerId</b>	<p>Metodo che restituisce l'Address che ha come customerId quello specificato.</p> <p><b>context</b> AddressDAO::findByCustomerId(customerId: Integer) <b>pre:</b></p>

	<p>customerId &gt; 0</p> <p><b>context</b> AddressDAO::findById(customerId: Integer) <b>post:</b> result.getCustomerId() = customerId</p>
--	---

CustomerOrderDAO	
<b>Description</b>	Classe DAO relativa alla classe CustomerOrder che consente di gestirne la persistenza.
<b>doSave</b>	<p>Metodo che permette di rendere persistente un CustomerOrder e ne restituisce il relativo id assegnato nel database. Inoltre, crea la relazione tra CustomerOrder e OrderProduct nel database.</p> <p><b>context</b> CustomerOrderDAO::doSave(order: CustomerOrder) <b>pre:</b> order &lt;&gt; null</p> <p><b>context</b> CustomerOrderDAO::doSave(order: CustomerOrder) <b>post:</b> this.findByKey(order.getId()) = order &amp; resi persistenti gli OrderProduct associati nel database</p>
<b>updateStatus</b>	<p>Metodo che permette di aggiornare lo stato di un ordine all'interno del database.</p> <p><b>context</b> CustomerOrderDAO::updateStatus(orderId: int, status: CustomerOrder.Status) <b>pre:</b> orderId &gt; 0 &amp; this.findByKey(orderId) &lt;&gt; null</p> <p><b>context</b> CustomerOrderDAO::updateStatus(orderId: int, status: CustomerOrder.Status) <b>post:</b> this.findByKey(orderId).getStatus() = status</p>
<b>findByKey</b>	<p>Metodo che restituisce il CustomerOrder che ha come id quello specificato.</p> <p><b>context</b> CustomerOrderDAO::findByKey(id: int) <b>pre:</b> id &gt; 0 &amp; CustomerOrder presente nel database</p> <p><b>context</b> CustomerOrderDAO::findByKey(id: int) <b>post:</b> this.findByKey(id) &lt;&gt; null</p>
<b>findByStatus</b>	<p>Metodo che restituisce tutti i CustomerOrder che hanno lo stato specificato.</p> <p><b>context</b> CustomerOrderDAO::findByStatus(status: CustomerOrder.Status) <b>pre:</b></p>



	<p>-</p> <p><b>context</b> CustomerOrderDAO::findByStatus(status: CustomerOrder.Status) <b>post:</b> result-&gt;forAll(co:CustomerOrder   co.getStatus() = status)</p>
<b>doRetrieveAll</b>	<p>Metodo che restituisce tutti i CustomerOrder salvati nel database.</p> <p><b>context</b> CustomerOrderDAO::doRetrieveAll() <b>pre:</b> -</p> <p><b>context</b> CustomerOrdeDAO::doRetrieveAll() <b>post:</b> -</p>

## PaymentManagement

PaymentAuthorizationServiceAdapter	
<b>Description</b>	Classe utilizzata per effettuare pagamenti al servizio esterno di pagamento.
<b>checkPaymentData</b>	<p>Metodo che controlla la validità dei dati di pagamento inseriti. In tal caso restituisce true, altrimenti false.</p> <p><b>context</b> PaymentAuthorizationServiceAdapter::checkPaymentData(cardNumber: String, expiration: LocalDate, cvv: String, price: Integer) <b>pre:</b> -</p> <p><b>context</b> PaymentAuthorizationServiceAdapter::checkPaymentData(cardNumber: String, expiration: LocalDate, cvv: String, price: Integer) <b>post:</b> cardNumber rispetta formato di soli numeri **** * **** * &amp; expiration &lt;&gt; null &amp; expiration valida &amp; cvv &lt;&gt; null &amp; cvv.length() = 3 &amp; cvv formato soli numeri &amp; price &gt;= 0</p>
<b>executePayment</b>	<p><b>context</b> PaymentAuthorizationServiceAdapter::executePayment(cardNumber: String, expiration: LocalDate, cvv: String, price: Integer) <b>pre:</b> this.checkPaymentData(cardNumber, expiration, cvv, price) = true</p>

	<b>context</b>
	PaymentAuthorizationServiceAdapter::executePayme
	nt(cardNumber: String, expiration: LocalDate, cvv:
	String, price: Integer) <b>post:</b>
	result = esito pagamento

## Object Design Model Optimization

Di seguito si riportano le modifiche effettuate al Class Diagram iniziale:

- La classe Order è stata rinominata in CustomerOrder per evitare ambiguità in termini di denominazione con altre classi di librerie Java. Inoltre vi è stato aggiunto l'attributo *id* per tenere traccia dell'identificativo univoco assegnato alla particolare istanza all'interno del database.
- Per una corretta e completa gestione dei prodotti, è stato aggiunto l'attributo *prodCode* alla classe Product. Tale attributo, permette di identificare un prodotto in base al suo codice univoco.
- Per semplificare la rimozione o l'aggiunta (se già presente nel sistema) di un prodotto al catalogo dei prodotti disponibili alla vendita è stato aggiunto l'attributo *disponibile*.
- Per i motivi riportati nei due punti precedenti sono stati aggiunti i metodi *makeProductAvailable* e *getProductByProdCode* nella classe ProductManager.
- E' stato aggiunto il metodo *getAllProducts* nella classe ProductManager per recuperare tutti i prodotti presenti nel sistema in ottica dello svolgimento del ruolo del ProductManager. Tale metodo si interfaccia con la classe ProductDAO.
- Nella classe Cart è stato rimosso il metodo *updateQuantity*. E' stato aggiunto, di conseguenza, il metodo *decreaseQuantity* che consente di diminuire la quantità di un prodotto nel carrello. Dell'incremento della quantità, invece, se ne occupa il metodo *addToCart* il quale, oltre all'aggiunta di un prodotto (se non presente), si occupa di incrementarne la quantità qualora esso fosse già presente nel carrello.
- Per facilitare la creazione di un'istanza di un CustomerOrder è stato implementato il metodo *makeCustomerOrder*.
- E' stata trasformata la classe associativa *ItemCart* della relazione *molti a molti* tra Cart e Product in un oggetto che ha due relazioni binarie: una con Cart e una con Product.
- E' stata trasformata la classe associativa *OrderLine* della relazione *molti a molti* tra Order (si fa riferimento al nome del Class Diagram non ristrutturato) e Product in un oggetto che ha due relazioni binarie: una con CustomerOrder e una con Product.

