Università degli studi di Salerno
Dipartimento di Informatica
Corso di Laurea in Informatica

Ingegneria del Software

Object Design Document

"Techblin"

.

Docente:
Andrea De Lucia

Studenti:

Nome	*Matricola*
Della Corte Giacomo	06078
Di Pierno Andrea	05724
Tuozzo Gabriele	05916

Anno Accademico: 2020/21

Sommario

Revisioni

Data	Versione	Descrizione	Autore
22/01/2021	1.0	Creazione documento ed inserimento paragrafi	Tuozzo Gabriele
22/01/2021	1.1	Stesura documento	Di Pierno Andrea Tuozzo Gabriele Della Corte Giacomo
04/02/2021	2.0	Revisione documento	Di Pierno Andrea Tuozzo Gabriele Della Corte Giacomo

1. Introduzione

In questa fase viene chiuso il gap tra oggetti di applicazione e componenti off-the-shelf dettagliando ciò che è stato individuato in fase di analisi dei requisiti, tramite l'individuazione di oggetti addizionali e il raffinamento degli oggetti esistenti. Inoltre, durante questa fase si prevede di effettuare scelte implementative per quanto riguarda il modello di analisi così da scegliere tra tutte le soluzioni possibili, quella che permette di avvicinarsi ai design goal prefissati.

1.1Object design trade-off

Trade-offs	Motivazione
Usability vs Functionality	Poiché il nostro sistema potrebbe essere utilizzato da utenti poco esperti, inserire un'interfaccia utente ricca di funzionalità potrebbe essere deleteria per l'usabilità e quindi per il tasso di conversione. A tal proposito, prediligiamo suddividere le funzionalità del sistema tra le varie sezioni, in modo da garantire una buona usabilità.
Security vs Development Cost	Poiché il nostro sistema tratta dati sensibili, prediligiamo la sicurezza, così da assicurare ai nostri utenti la protezione dei loro dati. Infatti sfruttiamo la sicurezza del protocollo HTTP (HTTPS) per cifrare le informazioni nelle sezioni critiche, ovvero quelle in cui si inseriscono/richiedono dati sensibili. Inoltre, sono stati applicati appositi filtri per evitare Hijacking ed SQL Injection.

1.2Interface documentation guidelines

1.2.1 Pagine HTML

1.2.1.1 Commenti

I commenti possono essere considerati una strategia per rendere il nostro codice più leggibile nei punti più significativi. Si tratta, infatti, di indicazioni significative, ma invisibili al browser, che inserite in punti specifici del documento ci permette di mantenere l'orientamento anche in file molto lunghi e complessi. La sintattica è la seguente:

```
<!-- questo è un commento -->
```

1.2.1.2 Istruzioni

Al browser bisogna sempre dire dove comincia e dove finisce il file html, e questo si ottiene con le semplici istruzioni: <HTML> e </HTML>. Inoltre, il file html contiene due parti. la testa (head) e il corpo (body) e, naturalmente, bisogna dire al browser dove iniziano e dove finiscono queste parti. Si fa così: <HEAD> </HEAD> e <BODY> </BODY>. Pertanto, un file html contiene sempre, obbligatoriamente queste istruzioni:

```
<HTML>
        <HEAD>
        </HEAD>
        <BODY>
        </BODY>
</HTML>
```

Normalmente ci si mette il titolo della pagina, che comparirà in quella striscia in alto nello schermo, al di sopra dei vari menù e tasti cliccabili. È possibile fare ciò usando i tags: <TITLE> </TITLE>. Esempio:

```
<HTML>
        <HEAD>
                <TITLE> Titolo </TITLE>
        </HEAD>
        <BODY>
        </BODY>
</HTML>
```

modi dall'utente sulla pagina web in uso (mouse, tastiera ecc.…). Il codice Javascript deve seguire le stesse convenzioni per il layout e i nomi del codice Java.

1.3 Definizioni, acronomi e abbreviazioni

1.4 Riferimenti

- Bernd Bruegge & Allen H. Dutoit, Object-Oriented Software Engineering Using UML, Patterns, and Java, 3rd edition.
- Slide del corso di Ingegneria del Software (Prof. De Lucia).
- Documento RAD.
- Documento SDD.

2. Packages

View

Nome	Descrizione
vetrina	Visualizzazione vetrina venditore
ricerca	Visualizzazione prodotti cercati
login	Form per il login
registrazione	Form per la registrazione
cart	Visualizzazione prodotti nel carrello
ordini	Visualizzazione ordini dell'acquirente
about	Visualizzazione info sul sito
cambioemail	Form per cambio e-mail
cambiopassword	Form per cambio password
checkout	Form inserimento info per completare l'ordine
confermaordine	Visualizzazione dati sull'ordine appena effettuato
contatti	Visualizzazione info dati di contatti
gestionecatalogo	Visualizzazione prodotti nel catalogo per modificarli o eliminarli
gestioneprodotto	Form modifica dati relativi al prodotto
header	View da includere nelle altre pagine contenente la barra di navigazione (utilizzata nelle altre View per non inserire codice ridondante)
impostazioni	Visualizzazione pagina relative alle impostazioni per modificare password o email. Nel caso dell'amministratore permette di accedere a dati statistici ed alle modifica al catalogo.
index	Visualizzazione dell'homepage
nonautorizzato	Visualizzazione di una pagina che ci indica che non siamo autorizzati a compiere un'azione
nuovoadmin	Form inserimento nuovo admin
nuovoprodotto	Form inserimento nuovo prodotto

privacy	Visualizzazione informativa sulla privacy
prodotto	Visualizzazione dei dettagli sul prodotto
productError	Visualizzazione errore quando non è possibile mostrare il prodotto
registrazioneConclusa	Visualizzazione di conferma procedura della registrazione
riepilogoOrdine	Visualizzazione di un ordine già effettuato in passato.
statistiche	Visualizzazione statistiche relative alle azioni compiute dall'utente
successo	Visualizzazione di una pagina generica che indica il successo di un'operazione.(Es. modifica prodotto avvenuta, aggiunta prodotto eseguita)

Control

Nome	Descrizione
AdminControl	Gestisce la creazione di un admin e il calcolo delle statistiche.
AjaxControl	Gestisce le chiamate Ajax
CartControl	Gestisce il carrello
LoginControl	Gestisce il login
LogoutControl	Gestisce il logout
OrderControl	Gestisce gli ordini
ProductControl	Gestisce i prodotti
RegistrationControl	Gestisce la registrazione
SearchControl	Gestisce la ricerca
SellerControl	Gestisce le azioni del venditore
UserControl	Gestisce le modifiche dei dati dell'utente

Model

Nome	Descrizione
AdminModelDM	Si interfaccia con il database per andare a svolgere le operazioni dell'amministratore
BuyerModelDM	Si interfaccia con il database per andare a svolgere le operazioni dell'acquirente
OrderModelDM	Si interfaccia con il database per andare a svolgere le operazioni sugli ordini
OrderProductModelDM	Si interfaccia con il database per andare a svolgere le operazioni riguardante i prodotti in un ordine
ProductModelDM	Si interfaccia con il database per andare a svolgere le operazioni sui prodotti
SellerModelDM	Si interfaccia con il database per andare a svolgere le operazioni sul venditore
SellerProductModelDM	Si interfaccia con il database per andare a svolgere le operazioni sui prodotti venduti da un determinato venditore

Entity

Nome	Descrizione
AdminBean	Contiene le informazioni dell'admin
BuyerBean	Contiene le informazioni dell'acquirente
OrderBean	Contiene le informazioni dell'ordine
ProductBean	Contiene le informazioni del prodotto
SellerBean	Contiene le informazioni del venditore
SellerProductBean	Contiene le informazioni dei prodotti venduti da un venditore
Cart	Contiene le informazioni relative ai prodotti nel carrello
OrderProductBean	Contiene le informazioni relative ai prodotti nell'ordine

3. Interfacce delle classi

Nome	**AdminBean**
Descrizione	Contiene le informazioni relative all'admin
Attributi	-email : String -nome: String -cognome: String -password: String -registrazione: String
Signature dei metodi	+String::getEmail() +String::getNome() +String::getCognome() +String::getPassword() +String::getRegistrazione() +void::setNome(newNome:String) +void::setCognome(newCognome:String) +void::setEmail(newEmail:String) +void::setPassword(newPassword:String) +void::setRegistrazione(newRegistrazione:String)
Pre-condizioni	**Context** void::setEmail(newEmail:String) • newEmail.matches (^([a-zA-Z0-9_\\-\\.]+)@([a-zA-Z0-9_\\-\\.]+)\\.([a-zA-Z]{2,5})$)
	Context void::setCognome(newCognome:String) • newCognome.matches(/^[A-Za-z] {1,20}$/)
	Context void::setNome(newNome:String) • newNome.matches(/^[A-Za-z] {1,20}$/)
	Context void::setPassword(newPassword:String) • newPassword.matches (/^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])[0-9a-zA-Z]{8,32}$/)
	Context void::setRegistrazione (newRegistrazione:String) • newRegistrazione != null and !newRegistrazione.isBlank()
	Context boolean::equals(other:Object) • other != null and other.getClass().getName() = "SellerBean"
Post-condizioni	**Context** void::setNome(newNome:String)

	• getNome() = newNome
	Context void::setCognome(newCognome:String) • getCognome () = newCognome
	Context void::setEmail (newEmail:String) • getEmail () = newEmail
	Context void::setPassword(newPassword:String) • getPassword () = newPassword
	Context void::setRegistrazione (newRegistrazione:String) • getRegistrazione() = newRegistrazione
Invarianti	nome != null and !nome.isBlank() and cognome != null and !cognome.isBlank() and email != null and !email.isBlank() and password != null and !password.isBlank() and registrazione != null and !registrazione.isBlank()

Nome	**AdminModelDM**
Descrizione	Si occupa delle operazione CRUD nel database relative all'entity AdminBean.
Attributi	
Signature dei metodi	+AdminBean::doRetreiveByKey(email:String) +Collection<AdminBean>::doRetrieveAll(order:String) +void::doSave(admin:AdminBean) +void::doUpdate(admin:AdminBean) +void::doUpdate(admin:AdminBean, oldemail:String) +void::doDelete(admin:AdminBean)
Pre-condizioni	**Context** AdminBean::doRetrieveByKey(email:String) • email.matches"^([a-zA-Z0-9_\\-\\.]+)@([a-zA-Z0-9_\\-\\.]+)\\.([a-zA-Z]{2,5})$")
	Context void::doSave(admin:AdminBean) • admin!=null • (doRetrieveByKey((admin.getEmail()))=-1)
	Context void::doUpdate(admin:AdminBean) • admin!=null • (doRetrieveByKey(admin.getEmail())).getEmail() = admin.getEmail()
	Context void::doDelete(admin:AdminBean) • admin!=null • (doRetrieveByKey(admin.getEmail())).getEmail() = admin.getEmail()
	Context void::doUpdate(admin:AdminBean,oldemail:String) • Admin!=null • (doRetrieveByKey(oldemail)))!=-1

Post-condizioni	**Context** AdminBean::doRetrieveByKey(email:String)
	• bean:AdminBean != null
	• Se la ricerca nel DB ha successo:
	  ○ bean.getEmail() = code
	• Altrimenti:
	  ○ bean.getEmail() = -1

	Context Collection<AdminBean>::doRetrieveAll(order:String) • admins: Collection<AdminBean> != null • if(order != null) admins viene ordinato
	Context void::doSave(admin:AdminBean) • doRetrieveByKey(admin.getEmail()) = admin

	Context void::doUpdate(admin:AdminBean) • doRetrieveByKey(admin.getEmail()) = admin
	Context void::doUpdate(admin:AdminBean,oldemail:String) • doRetrieveByKey(admin.getEmail()) = admin
	Context void::doDelete(admin:AdminBean) • (doRetrieveByKey(admin.getEmail())).getEmail() = -1
Invarianti	

Nome	**BuyerBean**
Descrizione	Contiene le informazioni relative all'acquirente
Attributi	-email : String -nome: String -cognome: String -password: String -registrazione: String
Signature dei metodi	+String::getEmail() +String::getNome() +String::getCognome() +String::getPassword() +String::getRegistrazione() +void::setNome(newNome:String) +void::setCognome(newCognome:String) +void::setEmail(newEmail:String) +void::setPassword(newPassword:String) +void::setRegistrazione(newRegistrazione:String)
Pre-condizioni	**Context** void::setEmail(newEmail:String) • newEmail.matches(^([a-zA-Z0-9_\\-\\.]+)@([a-zA-Z0-9_\\-\\.]+)\\.([a-zA-Z]{2,5})$)
	Context void::setCognome(newCognome:String) newCognome.matches(/^[A-Za-z] {1,20}$/)
	Context void::setNome(newNome:String) • newNome.matches(/^[A-Za-z] {1,20}$/)
	Context void::setPassword(newPassword:String) • newPassword.matches • (/^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])[0-9a-zA-Z]{8,32}$/)
	Context void::setRegistrazione (newRegistrazione:String) • newRegistrazione != null and !newRegistrazione.isBlank()
	Context boolean::equals(other:Object) • other != null and other.getClass().getName() = "SellerBean"
Post-condizioni	**Context** void::setNome(newNome:String) • getNome() = newNome

	Context void::setCognome(newCognome:String) • getCognome () = newCognome
	Context void::setEmail (newEmail:String) • getEmail () = newEmail
	Context void::setPassword(newPassword:String) • getPassword () = newPassword
	Context void::setRegistrazione (newRegistrazione:String) • getRegistrazione() = newRegistrazione
Invarianti	nome != null and !nome.isBlank() and cognome != null and !cognome.isBlank() and email != null and !email.isBlank() and password != null and !password.isBlank() and registrazione != null and !registrazione.isBlank()

Nome	**BuyerModelDM**
Descrizione	Si occupa delle operazione CRUD nel database relative all'entity BuyerBean.
Attributi	
Signature dei metodi	+BuyerBean::doRetrieveByKey(email:String) +Collection<BuyerBean>::doRetrieveAll(order:String) +void::doSave(buyer:BuyerBean) +void::doUpdate(buyer:BuyerBean) +void::doDelete(buyer:BuyerBean)
Pre-condizioni	**Context** BuyerBean::doRetrieveByKey(email:String) • email.mathes(^([a-zA-Z0-9_\\-\\.]+)@([a-zA-Z0-9_\\-\\.]+)\\.([a-zA-Z]{2,5})$)
	Context void::doSave(buyer:BuyerBean) • buyer!= null • (doRetrieveByKey(buyer.getEmail())).getEmail() = -1
	Context void::doUpdate(buyer:BuyerBean) • buyer!= null • (doRetrieveByKey(buyer.getEmail ())).getEmail() = buyer.getEmail()
	Context void::doDelete(buyer:BuyerBean) • buyer!= null • (doRetrieveByKey(buyer.getEmail())).getEmail() = seller.getEmail()

Post-codizioni	**Context** BuyerBean::doRetrieveByKey(email:String)
	• bean:BuyerBean != null
	• Se la ricerca nel DB ha successo:
	  ○ bean.getEmail() = code
	• Altrimenti:
	  ○ bean.getEmail() = -1

	Context Collection<BuyerBean>::doRetrieveAll(order:String) • buyer: Collection<BuyerBean> != null • if(order != null) buyer viene ordinato
	• **Context** void::doSave(buyer:BuyerBean) doRetrieveByKey(buyer.getEmail()) = buyer
	Context void::doUpdate(buyer:BuyerBean)

	doRetrieveByKey(buyer.getEmail()) = buyer
	Context void::doDelete(buyer:BuyerBean) • (doRetrieveByKey(buyer.getEmail())).getEmail() = -1
Invarianti	

Nome	**Cart**
Descrizione	Classe che realizza la funzione di carrello all'interno del Sistema.
Attributi	
Signature dei metodi	+addItem(item:T) +deleteItem(item:T) +List<T>::getItems() +deleteItems()
Pre-condizioni	**Context** Cart::addItem(item:T) • item!=null
	Context Cart::deleteItem(item:T) • item!=null
Post-condizioni	**Context** Cart::addItem(item:T) • getItems()=@pre.getItems()+1
	Context Cart::deleteItem(item:T) • getItems()=@pre.getItems()-1
	Context Cart::deleteItems() • getItems()=getItems()->isEmpty
Invarianti	

Nome	**OrderBean**
Descrizione	Contiene le informazioni relative all'ordine.
Attributi	-id:int -totale:int -dataOrdine:Date -stato:String -acquirente:String -indirizzo:String -cap:String -città:String -provincia:String -tipologiaPagamento:String -datiPagamento:String
Signature dei metodi	+int::getId() +int::getTotale() +Date::getDataOrdine() +String::getStato() +String::getAcquirente() +String::getIndirizzo() +String::getCap() +String::getCittà() +String::getProvincia() +String::getTipologiaPagamento() +String::getDatiPagamento() +void::setId(newId:int) + void::setTotale(newTotale:int) + void::setDataOrdine(newDataOrdine:Date) + void::setStato(newStato:String) + void::setAcquirente(newAcquirente:String) + void::setIndirizzo(newIndirizzo:String) + void::setCap(newCap:String) + void::setProvincia(newProvincia:String) + void::setCittà(newCittà:String) + void::setTipologiaPagamento(newTipologiaPagamento:String) + void::setDatiPagamento(newDatiPagamento:String)
Pre-condizioni	**Context** void::setId(newId:int) • newId != null and newId > 0
	Context void::setTotale(newTotale:int) • newTotale!=null and newTotale > 0
	Context void::setDataOrdine(newDataOrdine:Date) • newDataOrdine!=null
	Context void::setStato(newStato:String)

	• newStato!=null and !newStato.isBlank()
	Context void::setAcquirente(newAcquirente:String) • newAcquirente.matches(/^[A-Za-z] {5,50}$/)
	Context void::setIndirizzo(newIndirizzo:String) • newIndirizzo.matches(/^[a-zA-Z0-9,.]*$/)
	Context void::setCap(newCap:String) • newCap.matches(/^([0-9]{5})$/)
	Context void::setProvincia(newProvincia:String) • newProvincia!=null and ! newProvincia.isBlank()
	Context void::setCittà(newCittà:String) • newCittà.matches(/^[a-zA-Z]+(?:[\s-][a-zA-Z]+)*$/)
	Context void::setTipologiaPagamento(newTipologiaPagamento:String) • newTipologiaPagamento!=null and ! newTipologiaPagamento.isBlank()
	Context void::setDatiPagamento(newDatiPagamento:String) • newDatiPagamento!=null and ! newDatiPagamento.isBlank()
Post-condizioni	**Context** void::setId(id:int) • getId() = newId
	Context void::setTotale(newTotale:int) • getTotale() = newTotale
	Context void::setDataOrdine(newDataOrdine:Date) • getDataOrdine() = newDataOrdine
	Context void::setStato(newStato:String) • getStato() = newStato
	Context void::setAcquirente(newAcquirente:String) • getAcquirente() = newAcquirente
	Context void::setIndirizzo(newIndirizzo:String) • getIndirizzo() = newIndirizzo
	Context void::setCap(newCap:String) • getCap() = newCap
	Context void::setProvincia(newProvincia:String) • getProvincia() = newProvincia
	Context void::setCittà(newCittà:String) • getCittà() = newCittà
	Context void::setTipologiaPagamento(newTipologiaPagamento:String) • getTipologiaPagamento() = newTipologiaPagamento
	Context void::setDatiPagamento(newDatiPagamento:String) • getDatiPagamento() = newDatiPagamento
Invarianti	id>0 and id != null and totale != null and dataOrdine != null and stato != null and !stato.isBlank() and acquirente != null and !acquirente.isBlank() and indirizzo != null and !indirizzo.isBlank() and cap != null and !cap.isBlank() and provincia != null and !provincia.isBlank() and tipologiaPagamento != null and

	!tipologiaPagamento.isBlank() and datiPagamento != null and !datiPagamento.isBlank()

Nome	**OrderModelDM**
Descrizione	Si occupa delle operazione CRUD nel database relative all'entity OrderBean.
Attributi	
Signature dei metodi	+OrderBean::DoRetrieveByKey(id:int) +Collection<OrderBean>::doRetreiveAll(order:String) +doSave(order:String) +doUpdate(order:OrderBean) +doDelete(order:OrderBean)
Pre-condizioni	**Context** OrderBean::doRetrieveByKey(id:int) • Id != null and Id > 0
	Context void::doSave(order:OrderBean) • order != null • (doRetrieveByKey(order.getId())).getId() = -1
	Context void::doUpdate(order:OrderBean) • order != null • (doRetrieveByKey(order.getId ())).getId() = order.getId()
	Context void::doDelete(order:OrderBean) • order!= null • (doRetrieveByKey(order.getId ())).getId() = order.getId()

Post-condizioni	**Context** OrderBean::doRetrieveByKey(id:int)
	• bean:OrderBean != null
	• Se la ricerca nel DB ha successo:
	  ○ bean.getId() = id
	• Altrimenti:
	  ○ bean.getId() = -1

	Context Collection<OrderBean>::doRetrieveAll(order:String) • orders: Collection<OrderBean> != null • if(order != null) orders viene ordinato
	Context void::doSave(order:OrderBean) • doRetrieveByKey(order.getId()) = order

	Context void::doUpdate(order:OrderBean) doRetrieveByKey(order.getId()) = order
	Context void::doDelete(order:OrderBean) • (doRetrieveByKey(order.getId())).getId() = -1
Invarianti	

Nome	**OrderProductBean**
Descrizione	Rappresenta un' istanza della relazione fra prodotti e ordine.
Attributi	+prodotto:int +ordine:int
Signature dei metodi	+int::getProdotto() +int::getOrdine() +void::setProdotto(newProdotto:int) +void::setOrdine(newOrdine:int)
Pre-condizioni	**Context** void::setProdotto(newProdotto:int) • newProdotto != null and newProdotto > 0
	Context void::setProdotto(newOrdine:int) • newOrdine != null and newOrdine > 0
Post-condizioni	**Context** void::setProdotto(newProdotto:int) • getProdotto() = newProdotto
	Context void::setProdotto(newOrdine:int) • getOrdine() = newOrdine
Invarianti	prodotto>0 and prodotto != null and ordine>0 and ordine != null

Nome	**OrderProductModelDM**
Descrizione	Si occupa delle operazione CRUD nel database relative all'entity OrderProductBean.
Attributi	
Signature dei metodi	+OrderProductBean::doRetrieveByKey(prodotto:int, ordine:int) +Collection<OrderProductBean>::doRetrieveAll(order:String) +void::doSave(po:OrderProductBean) +void::doUpdate(po:OrderProductBean) +void::doDelete(po:OrderProductBean)
Pre-condizioni	**Context** OrderProductBean::doRetrieveByKey(prodotto:int, ordine:int) • prodotto != null • ordine != null
	Context void::doSave(po:OrderProductBean) • po != null • (doRetrieveByKey((po.getProdotto(), po.getOrdine())).getProdotto() = -1 and (doRetrieveByKey((po.getProdotto(), po.getOrdine())).getOrdine() = -1
	Context void::doUpdate(po:OrderProductBean) • po != null • (doRetrieveByKey((po.getProdotto(), po.getOrdine())).getProdotto() = po.getProdotto() and (doRetrieveByKey((po.getProdotto(), po.getOrdine())).getOrdine() = po.getOrdine()
	Context void::doDelete(po:OrderProductBean) • po != null • (doRetrieveByKey((po.getProdotto(), po.getOrdine())).getProdotto() = po.getProdotto() and (doRetrieveByKey((po.getProdotto(), po.getOrdine())).getOrdine() = po.getOrdine()
Post-condizioni	**Context** OrderProductBean::doRetrieveByKey(prodotto:int, ordine:int) • bean:OrderProductBean != null • Se la ricerca nel DB ha successo: ○ bean.getProdotto() = prodotto and bean.getOrdine() = ordine

	• Altrimenti: ○ bean.getProdotto() = -1 and bean.getOrdine() = -1
	Context Collection<OrderProductBean>::doRetrieveAll(order:String) • orderProducts: Collection<OrderProductBean> != null • if(order != null) orderProducts viene ordinato
	Context void::doSave(po:OrderProductBean) • doRetrieveByKey(po.getProdotto(), po.getOrdine()) = po
	Context void::doUpdate(po:OrderProductBean) • doRetrieveByKey(po.getProdotto(), po.getOrdine()) = po
	Context void::doDelete(po:OrderProductBean) • (doRetrieveByKey((po.getProdotto(), po.getOrdine())).getProdotto() = -1 and (doRetrieveByKey((po.getProdotto(), po.getOrdine())).getOrdine() = -1
Invarianti	

Nome	**ProductBean**
Descrizione	Contiene le informazioni relative ad un prodotto.
Attributi	code:int name:String description:String price:int quantity:int visible:int
Signature dei metodi	+int::getCode() +void::setCode(code:int) +String::getName() +void::setName(name:String) +String::getDescription() +void::setDescription(description:String) +int::getPrice() +void::setPrice(price:int) +int::getQuantity() +void::setQuantity(quantity:int) +int::getVisible() +void::setVisible() +void::setInvisible() +boolean::equals(other:Object) +String::toString()
Pre-condizioni	**Context** void::setCode(code:int) • code != null and code > 0

	Context void::setName(name:String) • name.matches(^.{5,50}$/)
	Context void::setDescription(description:String) • description.matches(/^.{10,150}$/)
	Context void::setPrice(price:int) • price.matches(/^[0-9]{1,7}$/)
	Context void::setQuantity(quantity:int) • quantity.matches(/^[0-9]{1,5}$/)
	Context boolean::equals(other:Object) • other != null and other.getClass().getName() = "ProductBean"
Post-condizioni	**Context** void::setCode(code:int) • getCode() = code
	Context void::setName(name:String) • getName() = name
	Context void::setDescription(description:String) • getDescription() = description
	Context void::setPrice(price:int) • getPrice() = price
	Context void::setQuantity(quantity:int) • getQuantity() = quantity
	Context void::setVisible() • getVisible() = 1

	Context void::setInvisible() • getVisible() = 0
Invarianti	code > 0 and code != null and name != null and !name.isBlank() and description != null and !description.isBlank() and price != null and price > 0 and quantity != null and quantity >= 0 and visible >= 0 and visible <= 1

Nome	**ProductModelDM**
Descrizione	Si occupa delle operazione CRUD nel database relative all'entity ProductBean.
Attributi	
Signature dei metodi	+ProductBean::doRetrieveByKey(code:String) +Collection<ProductBean>::doRetrieveAll(order:String) +void::doSave(product:ProductBean) +void::doUpdate(product:ProductBean) +void::doDelete(product:ProductBean)
Pre-condizioni	**Context** ProductBean::doRetrieveByKey(code:String) • code != null and !code.isBlank() and code>0
	Context void::doSave(product:ProductBean) • product != null • (doRetrieveByKey(product.getCode())).getCode() = -1
	Context void::doUpdate(product: ProductBean) • product != null • (doRetrieveByKey(product.getCode())).getCode() = product.getCode()
	Context void::doDelete(product:ProductBean) • product != null • (doRetrieveByKey(product.getCode())).getCode() = product.getCode()

Post-condizioni	**Context** ProductBean::doRetrieveByKey(code:String)
	• bean:ProductBean != null
	• Se la ricerca nel DB ha successo:
	  ○ bean.getCode() = code
	• Altrimenti:
	  ○ bean.getCode() = -1

	Context Collection<ProductBean>::doRetrieveAll(order:String) • products: Collection<ProductBean> != null • if(order != null) products viene ordinato
	Context void::doSave(product:ProductBean) • doRetrieveByKey(product.getCode()) = product
	Context void::doUpdate(product:ProductBean) • doRetrieveByKey(product.getCode()) = product
	Context void::doDelete(product:ProductBean) • (doRetrieveByKey(product.getCode())).getCode() = -1
Invarianti	

Nome	**SellerBean**
Descrizione	Contiene le informazioni relative al venditore.
Attributi	ID:String nome:String cognome:String email:String password:String registrazione:String
Signature dei metodi	+String::getID() +void::setID(newID:String) +String::getNome() +void::setNome(newNome:String) +String::getCognome() +void::setCognome(newCognome:String) +String::getEmail() +void::setEmail(newEmail:String) +String::getPassword() +void::setPassword(newPassword:String) +String::getRegistrazione() +void::setRegistrazione (newRegistrazione:String) +boolean::equals(other:Object) +String::toString()
Pre-condizioni	**Context** void::setID(newID:String) • newId.matches (/^[a-zA-Z0-9]{4,20}$/)
	Context void::setEmail(newEmail:String) • newEmail.matches(^([a-zA-Z0-9_\\-\\.]+)@([a-zA-Z0-9_\\-\\.]+)\\.([a-zA-Z]{2,5})$)

	Context void::setCognome(newCognome:String) • newCognome.matches(/^[A-Za-z] {1,20}$/)
	Context void::setNome(newNome:String) • newNome.matches(/^[A-Za-z] {1,20}$/)
	Context void::setPassword(newPassword:String) • newPassword.matches • (/^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])[0-9a-zA-Z]{5,32}$/)
	Context void::setRegistrazione (newRegistrazione:String) • newRegistrazione != null and !newRegistrazione.isBlank()
	Context boolean::equals(other:Object) • other != null and other.getClass().getName() = “SellerBean”
Post-condizioni	**Context** void::setID(newID:String) • getID() = newID
	Context void::setNome(newNome:String) • getNome() = newNome
	Context void::setCognome(newCognome:String) • getCognome () = newCognome
	Context void::setEmail (newEmail:String) • getEmail () = newEmail
	Context void::setPassword(newPassword:String) • getPassword () = newPassword
	Context void::setRegistrazione (newRegistrazione:String) • getRegistrazione() = newRegistrazione

Invarianti	ID > 0 and ID != null and nome != null and !nome.isBlank() and cognome != null and !cognome.isBlank() and email != null and !email.isBlank() and password != null and !password.isBlank() and registrazione != null and !registrazione.isBlank()

Nome	**SellerModelDM**
Descrizione	Si occupa delle operazione CRUD nel database relative all'entity SellerBean.
Attributi	
Signature dei metodi	+SellerBean::doRetrieveByKey(email:String) +Collection<SellerBean>::doRetrieveAll(order:String) +void::doSave(seller:SellerBean) +void::doUpdate(seller:SellerBean) +void::doDelete(seller:SellerBean)
Pre-condizioni	**Context** SellerBean::doRetrieveByKey(email:String) • email.matches(^([a-zA-Z0-9_\\-\\.]+)@([a-zA-Z0-9_\\-\\.]+)\\.([a-zA-Z]{2,5})$)
	Context void::doSave(seller:SellerBean) • seller!= null • (doRetrieveByKey(seller.getEmail())).getEmail() = -1
	Context void::doUpdate(seller:SellerBean) • seller!= null • (doRetrieveByKey(seller.getEmail ())).getEmail() = seller.getEmail()
	• **Context** void::doDelete(seller:SellerBean) • product != null • (doRetrieveByKey(seller.getEmail())).getEmail() = seller.getEmail()

Post-condizioni	**Context** SellerBean::doRetrieveByKey(email:String)
	• bean:SellerBean != null
	• Se la ricerca nel DB ha successo:
	  ○ bean.getEmail() = code
	• Altrimenti:
	  ○ bean.getEmail() = -1

	Context Collection<SellerBean>::doRetrieveAll(order:String) • sellers: Collection<SellerBean> != null • if(order != null) sellers viene ordinato
	• **Context** void::doSave(seller:SellerBean) • doRetrieveByKey(product.getCode()) = seller
	Context void::doUpdate(seller:SellerBean) • doRetrieveByKey(seller.getEmail()) = seller
	Context void::doDelete(seller:SellerBean) • (doRetrieveByKey(seller.getEmail())).getEmail() = -1
Invarianti	

Nome	**SellerProductBean**
Descrizione	Rappresenta una istanza della relazione tra venditore e prodotto.
Attributi	prodotto:int venditore:String
Signature dei metodi	+int::getProdotto() +void::setProdotto(newProdotto:int) +String::getVenditore() +void::setVenditore(newVenditore:String)
Pre-condizioni	**Context** void::setProdotto(newProdotto:int) • newProdotto != null and newProdotto > 0
	Context void::setVenditore(newVenditore:String) • newVenditore .matches((^([a-zA-Z0-9_\\-\\.]+)@([a-zA-Z0-9_\\-\\.]+)\\.([a-zA-Z]{2,5})$))
Post-condizioni	**Context** int::getProdotto() • getID() = newID
	Context void::setProdotto(newProdotto:int) • getNome() = newNome
	Context String::getVenditore() • getCognome () = newCognome
	Context void::setVenditore(newVenditore:String) • getVenditore () = newVenditore
Invarianti	prodotto != null and prodotto > 0 and venditore != null and !venditore.isBlank()

Nome	**SellerProductModelDM**
Descrizione	Si occupa delle operazione CRUD nel database relative all'entity SellerProductBean.
Attributi	
Signature dei metodi	+SellerProductBean::doRetrieveByKey(venditore:String, prodotto:int) +Collection<SellerProductBean>::doRetrieveAll(order:String) +void::doSave(pmb:SellerProductBean) +void::doUpdate(pmb:SellerProductBean) +void::doDelete(pmb:SellerProductBean)
Pre-condizioni	**Context** SellerProductBean::doRetrieveByKey(venditore:String, prodotto:int) • venditore.matches(^([a-zA-Z0-9_\\-\\.]+)@([a-zA-Z0-9_\\-\\.]+)\\.([a-zA-Z]{2,5})$) • prodotto != null and prodotto > 0
	Context void::doSave(pmb:SellerProductBean) • pmb!= null • (doRetrieveByKey(pmb.getVenditore(),pmb.getProdotto())).getVenditore() = "" and (doRetrieveByKey(pmb.getVenditore(),pmb.getProdotto())).getProdotto() = -1
	Context void::doUpdate(pmb:SellerProductBean) • seller!= null • (doRetrieveByKey(pmb.getVenditore(),pmb.getProdotto())).getVenditore() = pmb.getVenditore() and (doRetrieveByKey(pmb.getVenditore(),pmb.getProdotto())).getProdotto() = pmb.getProdotto()
	Context void::doDelete(pmb:SellerProductBean) • product != null • (doRetrieveByKey(pmb.getVenditore(),pmb.getProdotto())).getVenditore() = pmb.getVenditore() and (doRetrieveByKey(pmb.getVenditore(),pmb.getProdotto())).getProdotto() = pmb.getProdotto()

Post-condizioni	**Context** SellerProductBean::doRetrieveByKey(venditore:String, prodotto:int)
	• bean:SellerProductBean!= null
	• Se la ricerca nel DB ha successo:
	  o bean.getVenditore() = venditore and bean.getProdotto() = prodotto
	• Altrimenti:
	  o bean.getVenditore() = “” and bean.getProdotto() = -1

	Context Collection<SellerProductBean>::doRetrieveAll(order:String) • sm: Collection< SellerProductBean > != null • if(order != null) sellers viene ordinato
	• **Context** void::doSave(pmb:SellerProductBean) • doRetrieveByKey(pmb.getVenditore(), pmb.getProdotto()) = pmb
	Context void::doUpdate(pmb:SellerProductBean) • doRetrieveByKey(pmb.getVenditore(), pmb.getProdotto()) = pmb
	Context void::doDelete(pmb:SellerProductBean) • (doRetrieveByKey(pmb.getVenditore(),pmb.getProdotto())).getVenditore() = “” and (doRetrieveByKey(pmb.getVenditore(),pmb.getProdotto())).getProdotto() = -1
Invarianti	

Nome	**StatsModelDM**
Descrizione	Si occupa del calcolo delle statistiche, andando a recuperare i dati nel database relativi ai prodotti, agli ordini e agli utenti registrati sul sito.
Attributi	
Signature dei metodi	+ArrayList<Integer>::calculate(from:Date, to:Date)
Pre-condizioni	**Context** ArrayList<Integer>::calculate(from:Date, to:Date) • from != null • to != null • from <= to (precedent or same date)
Post-condizioni	**Context** ArrayList<Integer>::calculate(from:Date, to:Date) • result:ArrayList<Integer>!=null
Invarianti	