

# iHotel

Progetto di ingegneria del software

Carocci Eugenio Cianetti Gabriele Ranalli Alessandro

A.A. 2013 / 2014

# Sommario

- Processo di sviluppo
- Ideazione
- Elaborazione
  - Iterazione 1
  - Iterazione 2
  - Iterazione 3
- Conclusioni

# Sommario

- Processo di sviluppo
- Ideazione
- Elaborazione
  - Iterazione 1
  - Iterazione 2
  - Iterazione 3
- Conclusioni

# Caratteristiche

Adozione del processo di sviluppo iterativo, evolutivo ed incrementale, **AUP**, le cui caratteristiche salienti sono:

- Affrontare le problematiche di rischio maggiore nelle iterazioni iniziali.
- Utilizzo di iterazioni “timeboxed”.
- Coinvolgimento continuo di progettisti ed utenti per valutazioni e testing in modo da ottenere feedback.
- Produzione di documentazione solo se realmente utile.
- Modellazione visuale UML.
- Utilizzo dei casi d’uso per la scoperta di requisiti del sistema.

# Fasi del processo di sviluppo - 1

## Ideazione:

- visione approssimativa, studio economico, stime approssimative di tempi e costi.

## Elaborazione:

- visione raffinata, implementazione iterativa, risoluzione dei rischi maggiori e identificazione della maggior parte dei requisiti.

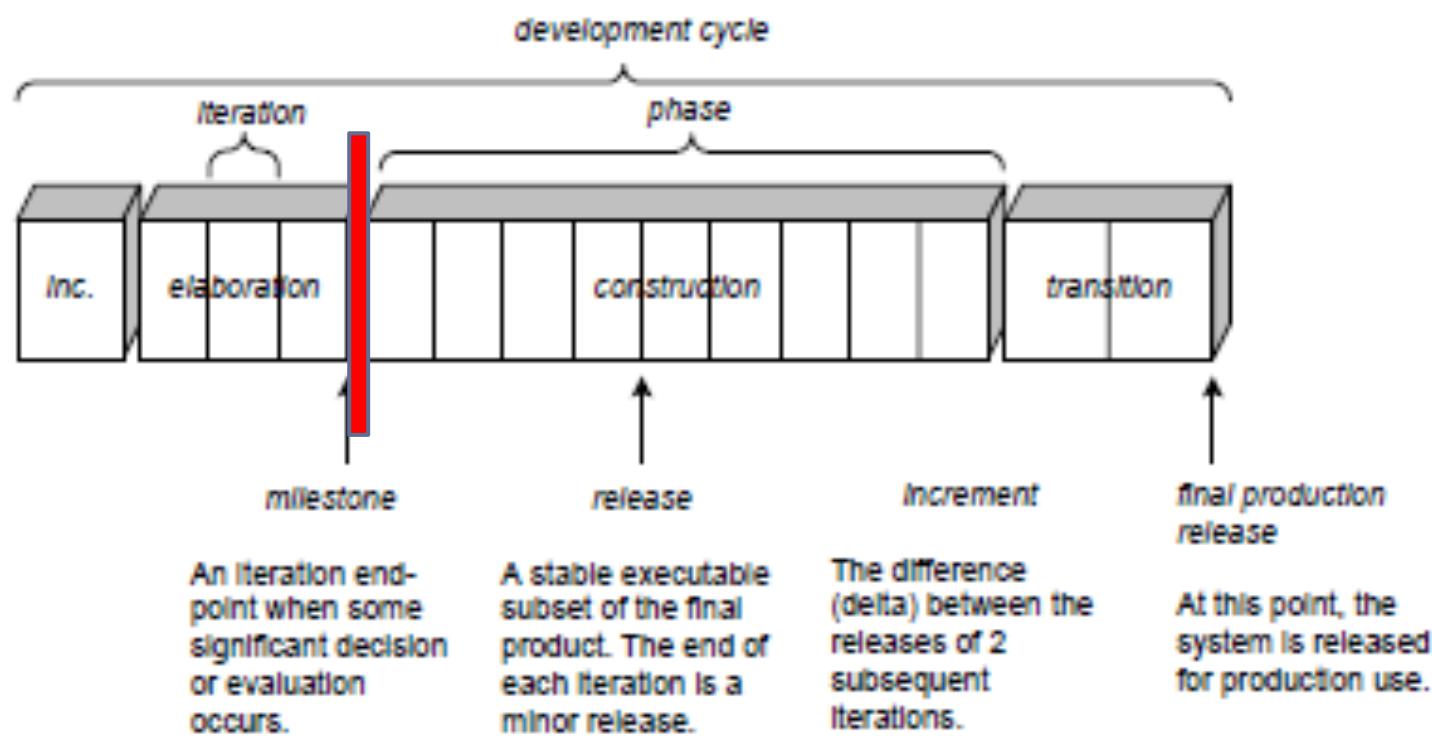
## Costruzione:

- implementazione iterativa degli elementi rimanenti, quelli con rischio minore.

## Transizione:

- beta test e rilascio.

# Fasi del processo di sviluppo - 2



# Sommario

- Processo di sviluppo
- Ideazione
- Elaborazione
  - Iterazione 1
  - Iterazione 2
  - Iterazione 3
- Conclusioni

# Obiettivi

- Stabilire una visione sugli obiettivi del progetto.
- Analizzare in modo leggero il dominio in cui ci si sta muovendo, avvalendosi anche di consulenze con esperti del settore.
- Individuare “approssimativamente” i requisiti del sistema e le criticità più evidenti.
- Formalizzare le informazioni raccolte in elaborati utili alle fasi successive.
  - Visione.
  - Specifiche supplementari.
  - Regole di dominio.
  - Modello dei casi d'uso.
  - Glossario.
  - ...

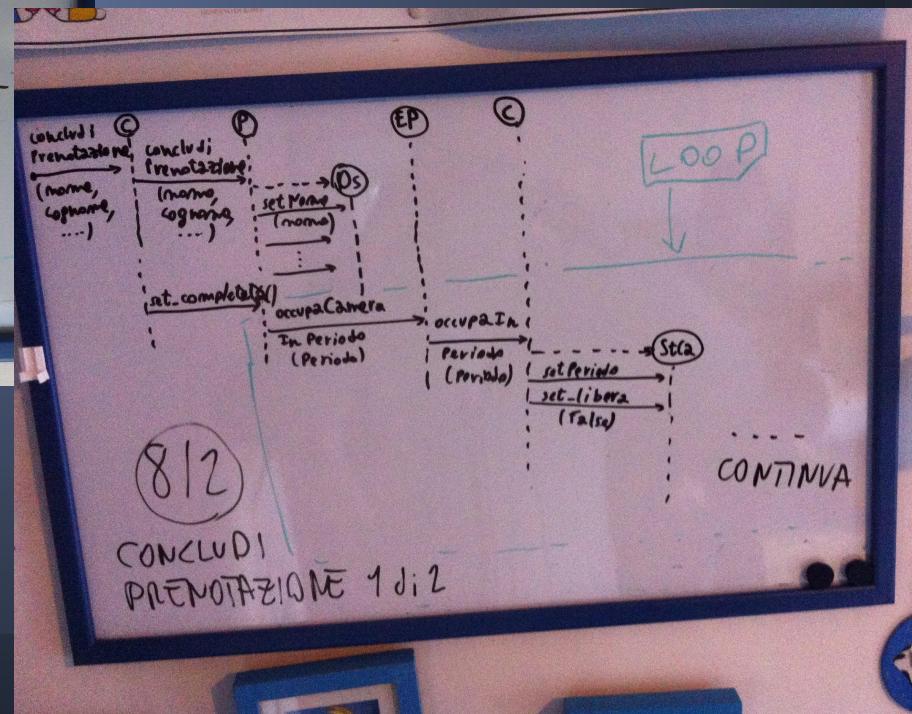
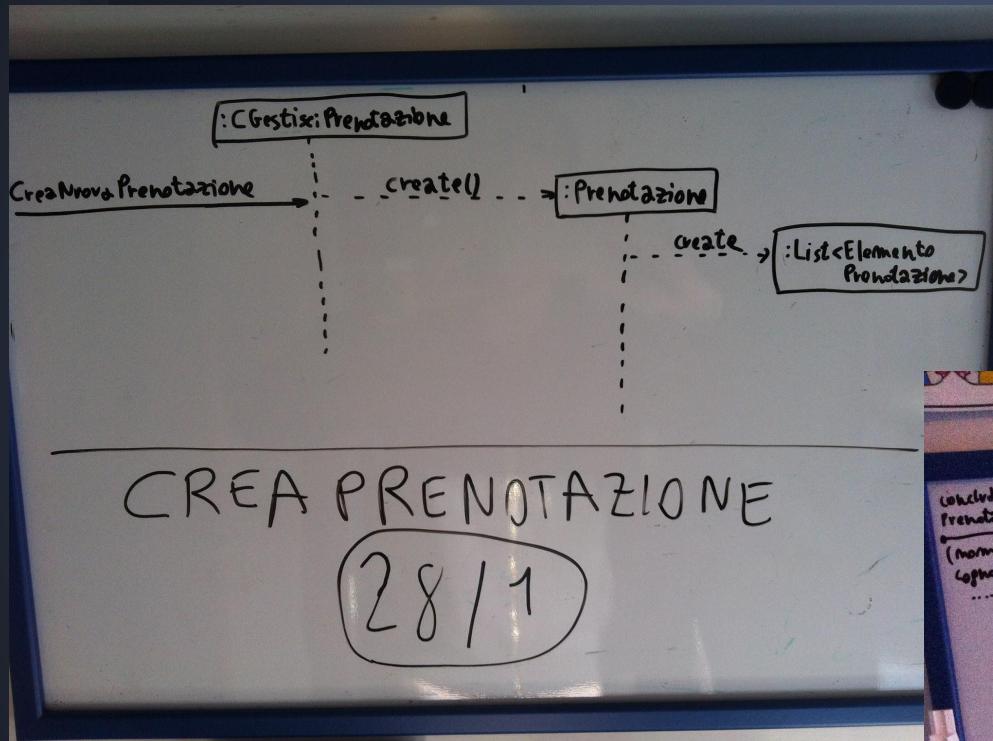
# Sommario

- Processo di sviluppo
- Ideazione
- Elaborazione
  - Iterazione 1
  - Iterazione 2
  - Iterazione 3
- Conclusioni

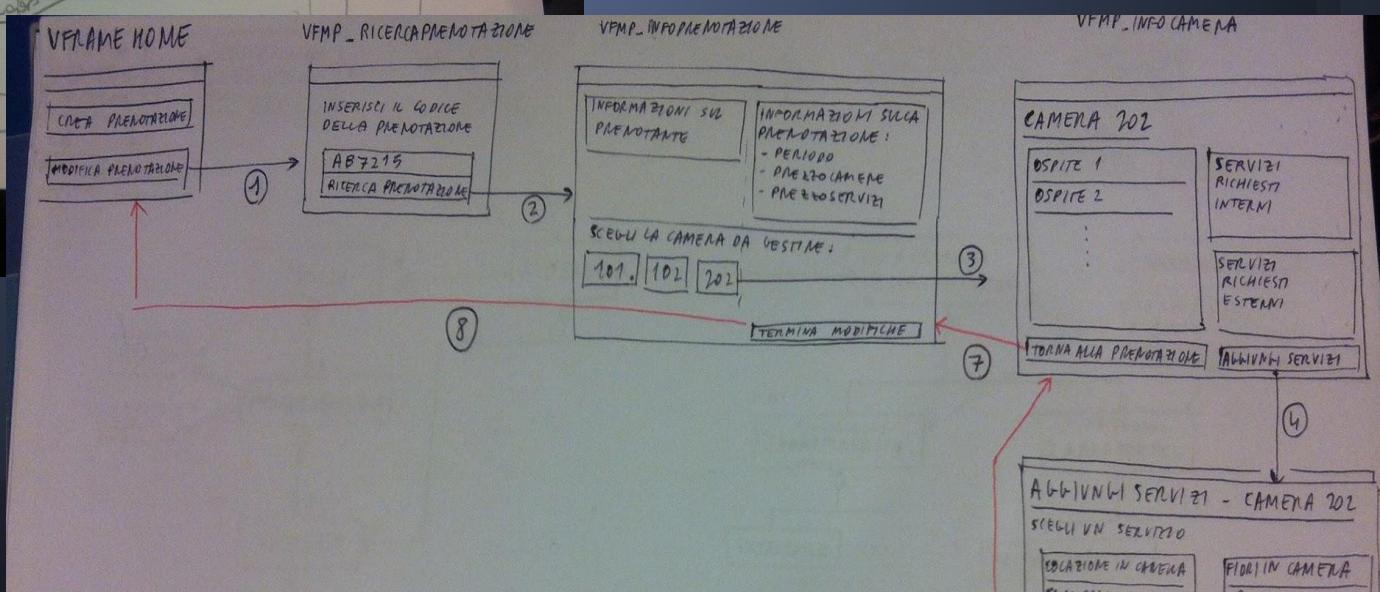
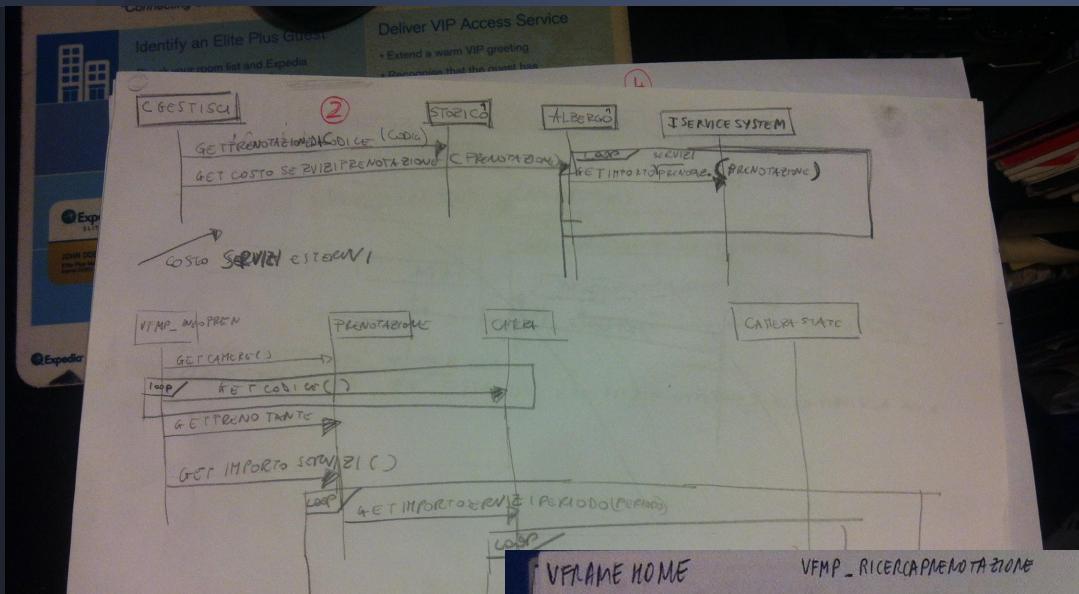
# Flusso della generica iterazione

- Revisione del caso del uso scelto per l' iterazione corrente.
- Schematizzazione delle interazioni con il sistema, mediante SSD.
- Analisi degli oggetti concettuali introdotti.
- Sviluppo di SD e progettazione di oggetti software coinvolgendo:
  - Principi grasp.
  - Design patterns.
- Codifica.
- Reverse engineering per generare la documentazione.
- Rilascio di un applicativo eseguibile.
- Brainstorming e descrizione dettagliata di uno o più casi d'uso, se necessaria.

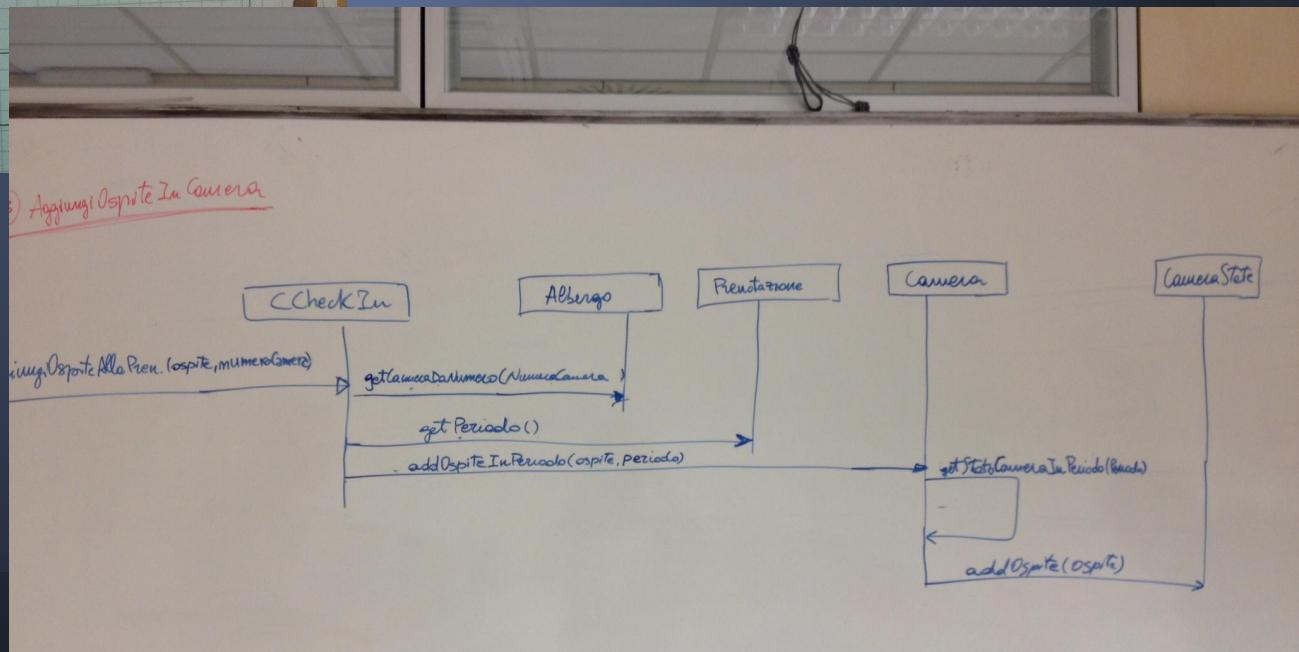
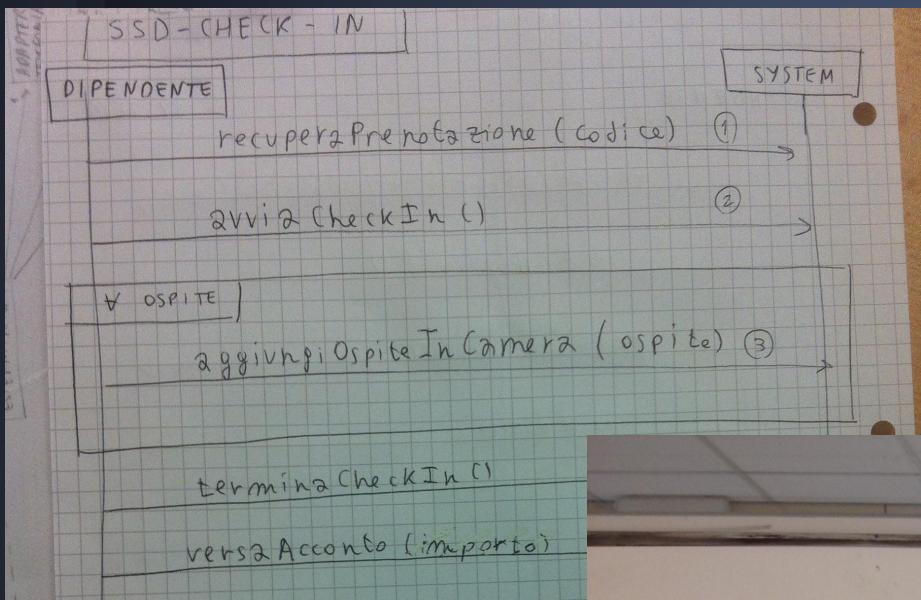
# Esempi di documentazione - 1



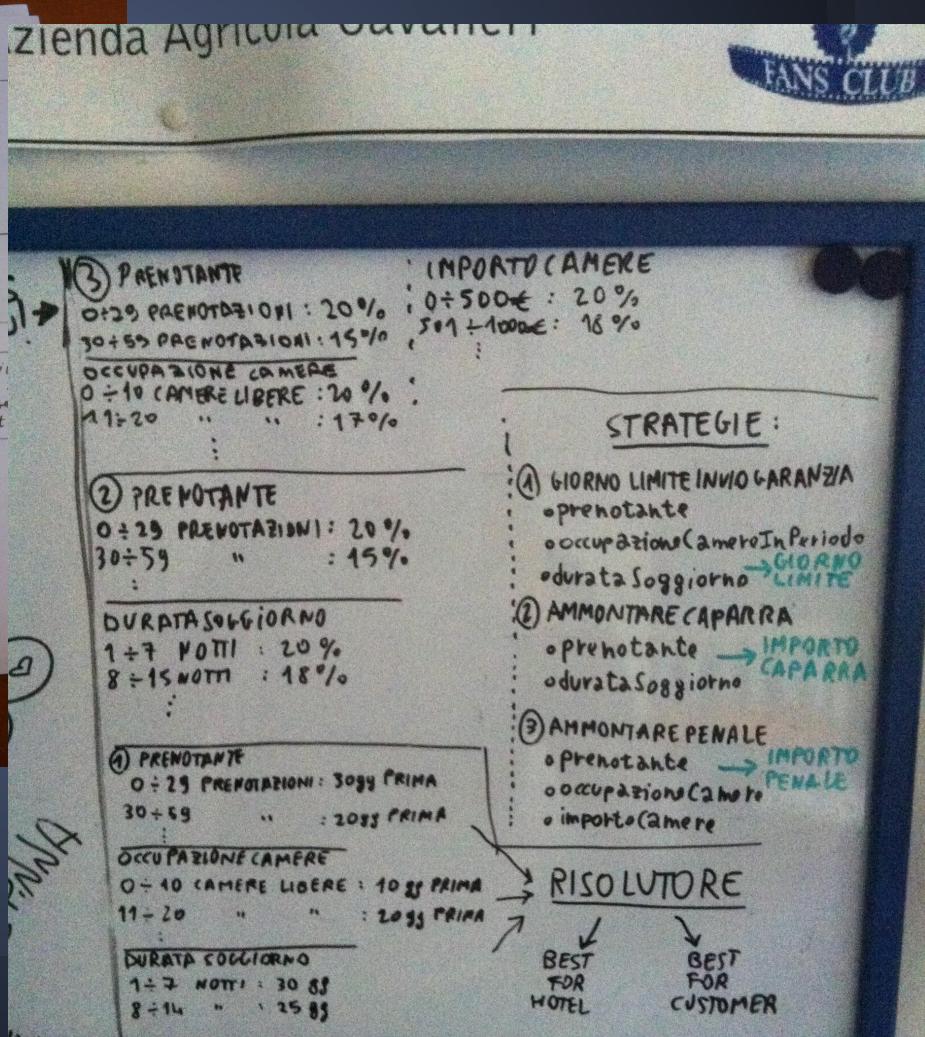
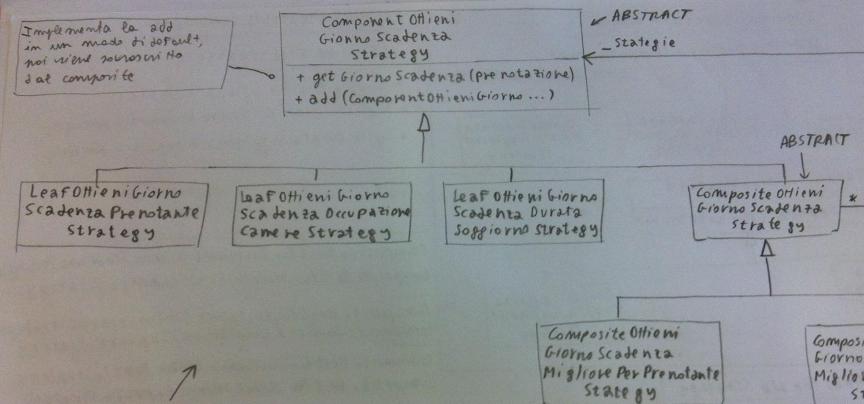
# Esempi di documentazione - 2



# Esempi di documentazione - 3



# Esempi di documentazione - 4



# Esempi di documentazione - 5



# Sommario

- Processo di sviluppo
- Ideazione
- Elaborazione
  - Iterazione 1
  - Iterazione 2
  - Iterazione 3
- Conclusioni

# Obiettivi

- Implementare lo scenario di successo per il caso d'uso “*Crea Prenotazione*”.
- Implementare un caso d'uso di “*Start up*”, necessario per gestire le esigenze di inizializzazione per questa iterazione.
- Realizzazione della base dati con tecnologia *ORM*.

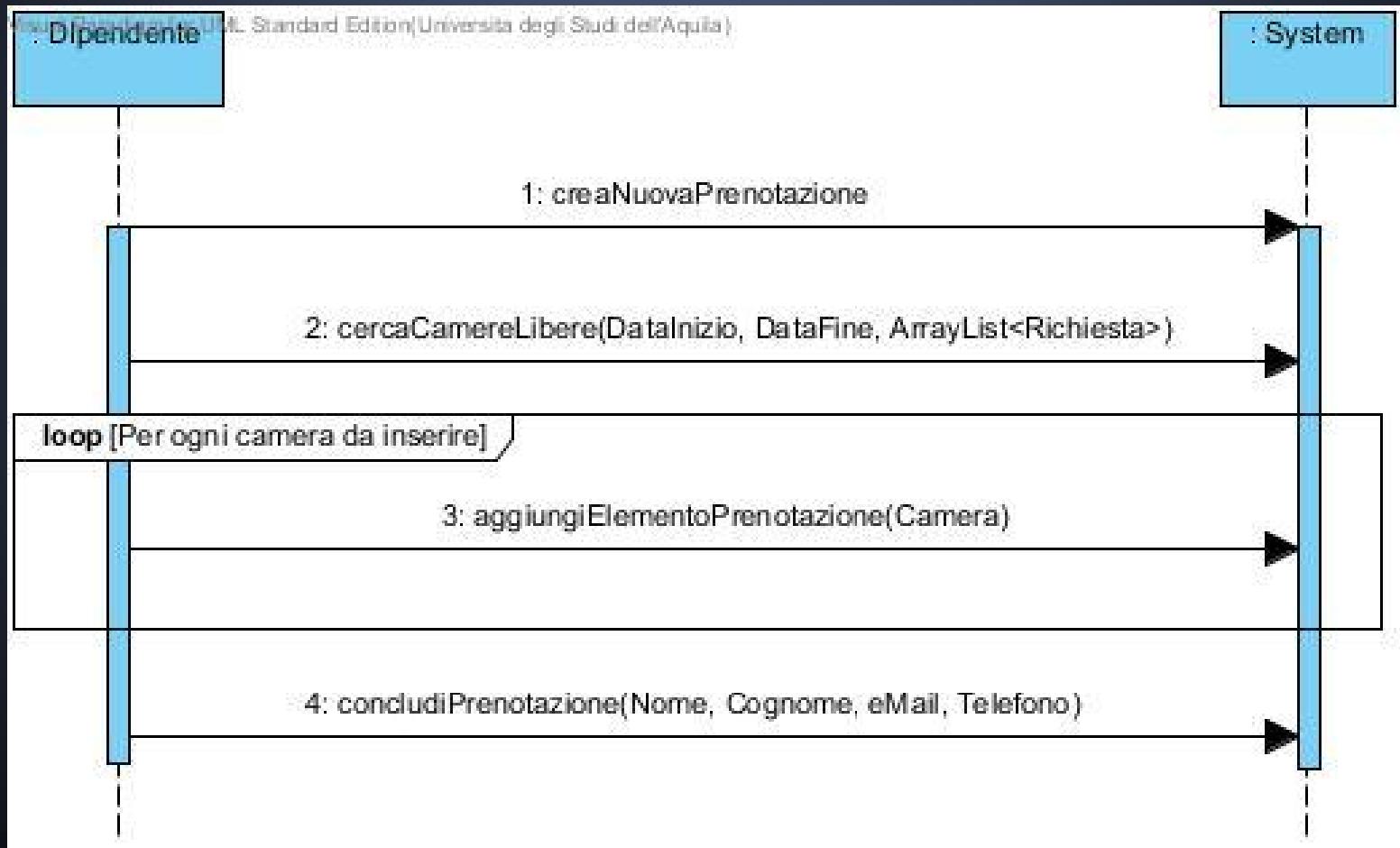
# Caso d'uso - Crea Prenotazione

## *Scenario di successo - 1*

1. Il cliente invia una e-mail con le informazioni necessarie per effettuare una prenotazione.
2. Il dipendente indica al sistema che vuole avviare la procedura per la creazione di una prenotazione.
3. Il dipendente inserisce il periodo della prenotazione e le tipologie di camere.
4. Il sistema mostra le soluzioni per le camere richieste.
5. Il dipendente sceglie una camera tra quelle proposte.
6. Il sistema mostra il prezzo totale della prenotazione.  
*Ripetere passo 6 per ogni camera che si vuole inserire.*
7. Il dipendente inserisce i dati relativi al cliente prenotante.
8. Il dipendente indica la fine della procedura di creazione.
9. Il sistema registra la nuova prenotazione.

# Caso d'uso - Crea Prenotazione

## *Scenario di successo - 2*



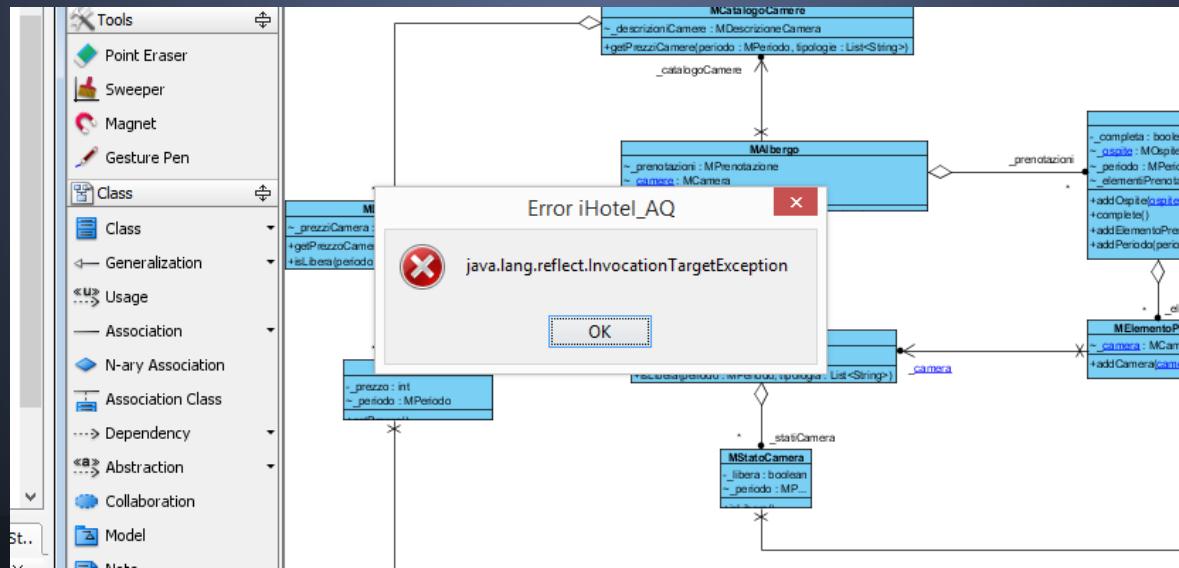
# Problematiche progettuali

- Concepimento di un buon modello di dominio.
- Definizione di contratti delle operazioni errati, che hanno portato a SD non corretti.
- Assegnazione delle responsabilità agli oggetti, mediante principi GRASP.

# Problematiche tecniche

## *Relazione con gli strumenti*

- Utilizzo di Visual Paradigm, nella creazione di SD, e nell'organizzazione dei contenuti.
- Errore tra Visual Paradigm e server SVN, in fase di commit.



# Problematiche tecniche

## *Strato di persistenza*

- Problematiche relative all'utilizzo della tecnologia **ORM** per la realizzazione e gestione della base dati, mediante l'utilizzo di Visual Paradigm.



- Si è passati ad una base dati ad oggetti, in particolare a **db4o**, per la quale è stata necessaria una breve fase di studio della letteratura a riguardo.

# Obiettivi proposti ...

- Collegamento con sistemi esterni, in particolare, con i channel manager, per la ricezione di nuove prenotazioni.
- Introduzione alle politiche di sconto, per la determinazione dei prezzi per la prenotazione.
- Un progetto per aggiornare una finestra della GUI quando cambia il totale della prenotazione, senza violare il *Principio di separazione Comando-Interrogazione*.

# ... e obiettivi scelti

In seguito al feedback ricevuto da un esperto di software design, si è deciso di sviluppare gli obiettivi seguenti:

- Risoluzione delle problematiche progettuali evidenziate.
- Realizzazione del caso d'uso “**Modifica Prenotazione**”, per accrescere la conoscenza del dominio.
- In particolare si è deciso di realizzare l'aggiunta di un servizio alla prenotazione.

# Sommario

- Processo di sviluppo
- Ideazione
- Elaborazione
  - Iterazione 1
  - Iterazione 2
  - Iterazione 3
- Conclusioni

# Problematiche progettuali

## *1 iterazione*

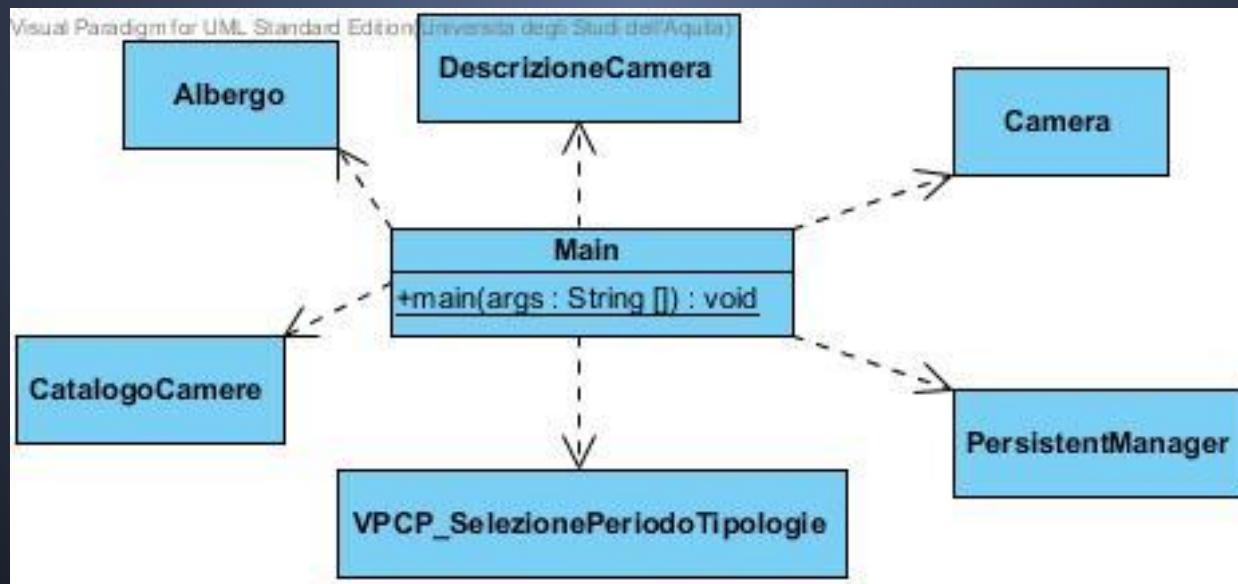
- Sovraccarico di responsabilità per la classe *Main*.
- Cattiva progettazione dello strato di persistenza.
- Bassa flessibilità nelle classi del package View.
- Assenza di separazione dalla logica della gestione di eventi, alla realizzazione della struttura dei Frame.
- Violazione del principio “Comando-Interrogazione”.
- Cattiva progettazione della logica per l’occupazione di camere a seguito di una prenotazione.

# Soluzioni ai problemi - 1 iterazione

## *Sovraccarico di responsabilità classe Main - 1*

Prima:

- Bassa coesione e alto accoppiamento.
- 35 SLOC.

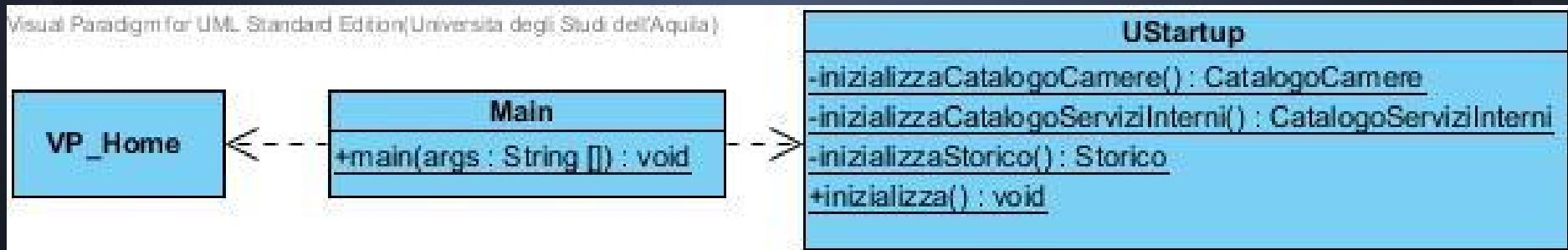


# Soluzioni ai problemi - 1 iterazione

## *Sovraccarico di responsabilità classe Main - 2*

Dopo:

- Progettazione di una classe con la responsabilità di inizializzare lo strato di dominio, *UStartup*.
- Progettazione di un Frame iniziale per l'applicazione.
- Grazie alle classi introdotte si è ottenuta una riduzione del codice a 5 SLOC.



# Soluzioni ai problemi - 1 iterazione

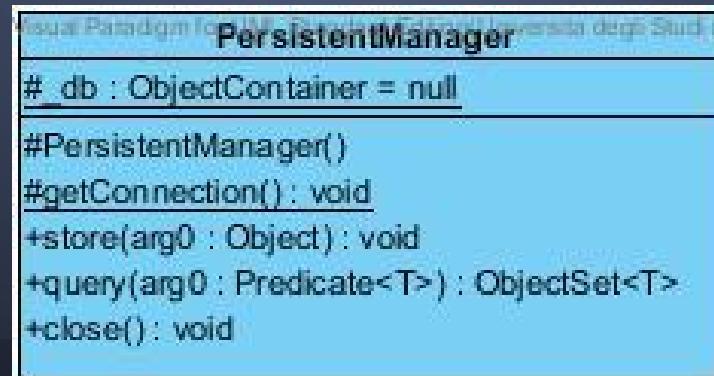
## *Cattiva progettazione strato Persistenza - 1*

Prima:

- Query create al di fuori del package Persistence.

Soluzione semplice:

- Far gestire le query all'oggetto *PersistentManager*.
- Bassa Coesione e alto accoppiamento.

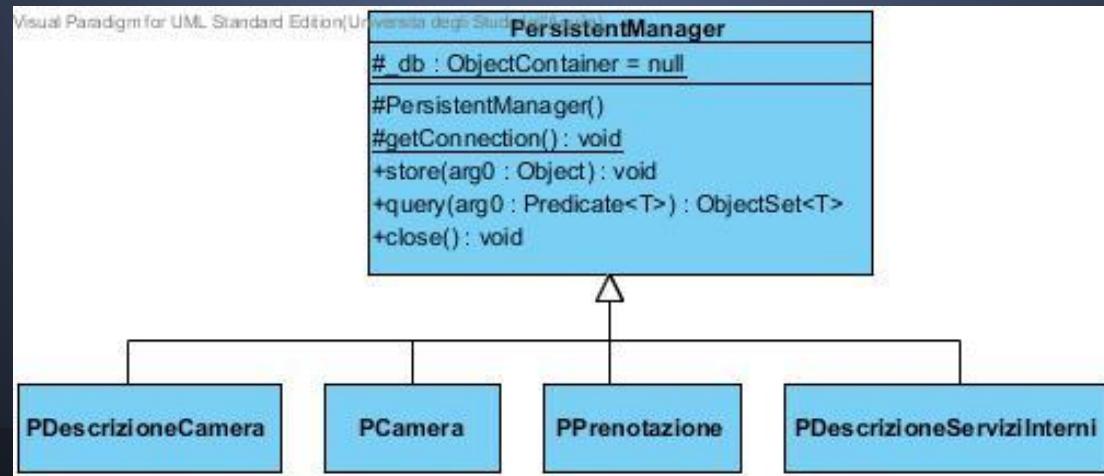


# Soluzioni ai problemi - 1 iterazione

## *Cattiva progettazione strato Persistenza - 2*

Dopo:

- Progettazione di classi ad-hoc nel package Persistence.
- Ogni classe di questo package, è responsabile per le operazioni di persistenza, per una particolare classe del package Model.



# Soluzioni ai problemi - 1 iterazione

## *Bassa flessibilità nelle classi del package View - 1*

Prima:

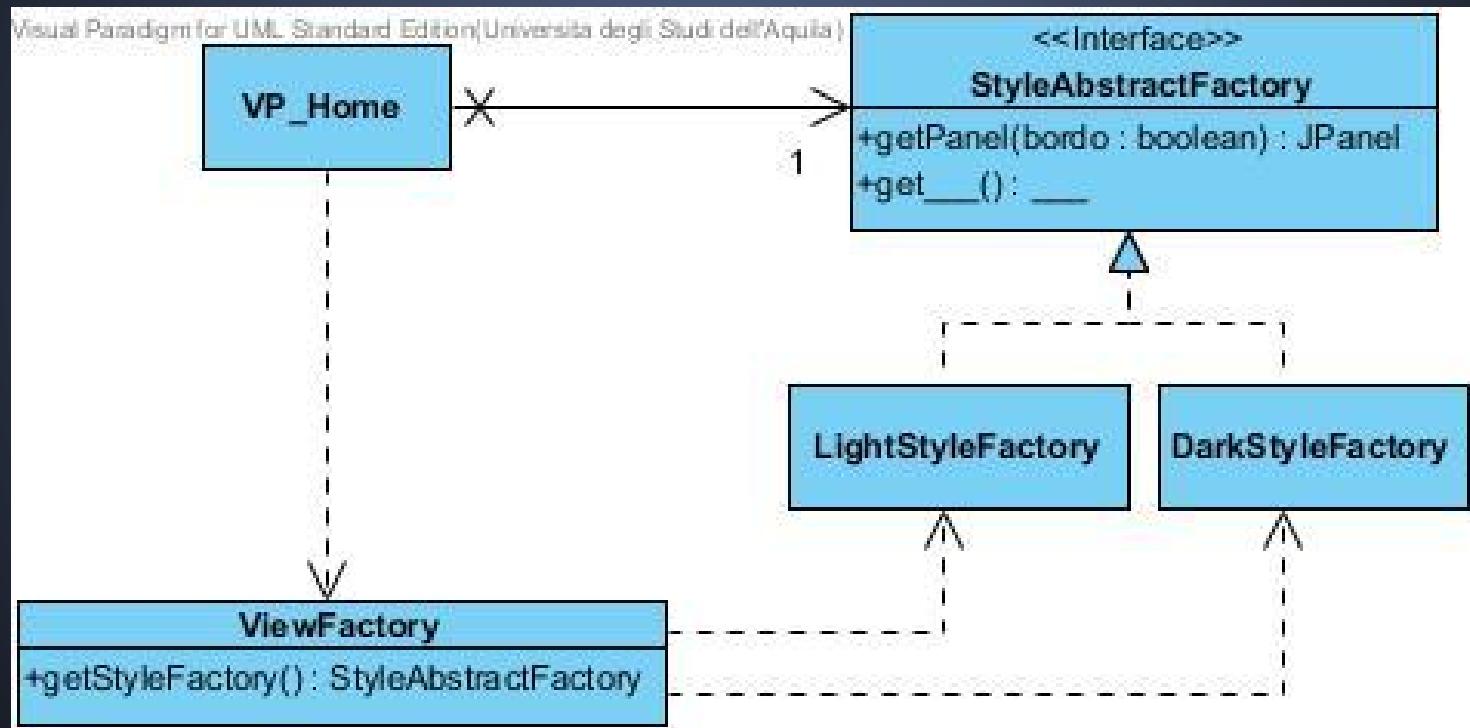
- Creazione diretta degli oggetti grafici.
- Legame diretto con lo specifico oggetto grafico.

Dopo:

- Applicazione del Design Pattern “**Abstract Factory**”.
- Progettazione di classi ad-hoc per la creazione di oggetti grafici.
- Variazione del “look & feel” secondo un parametro in un file di configurazione.
- Progettazione di una classe secondo il pattern “**Simple Factory**” con la responsabilità di fornire la corretta *concrete factory*.

# Soluzioni ai problemi - 1 iterazione

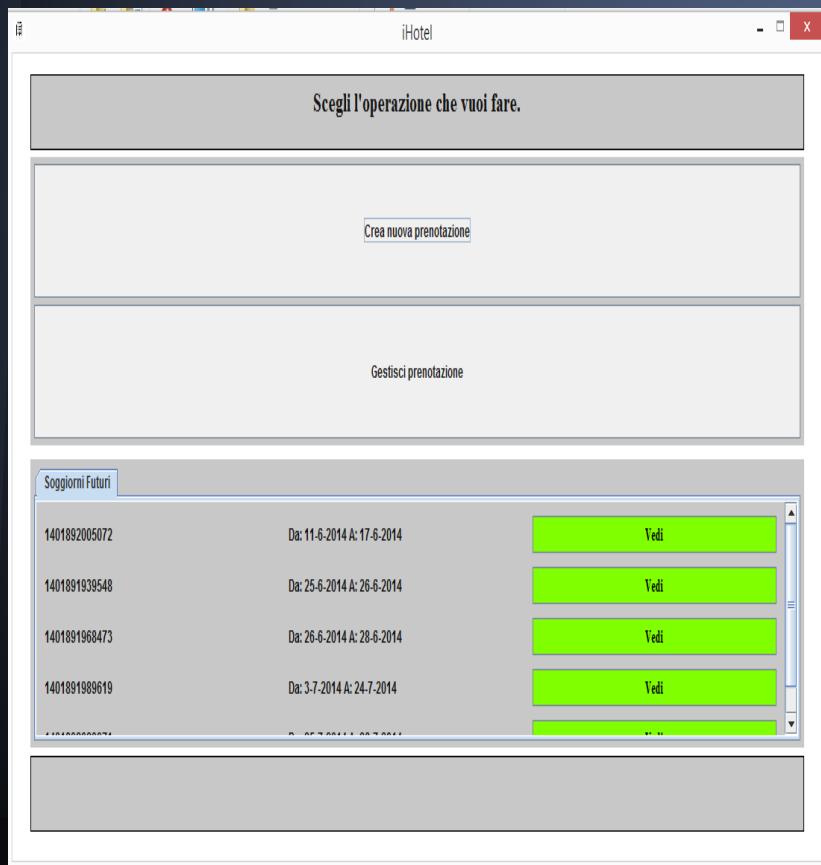
*Bassa flessibilità nelle classi del package View - 2*



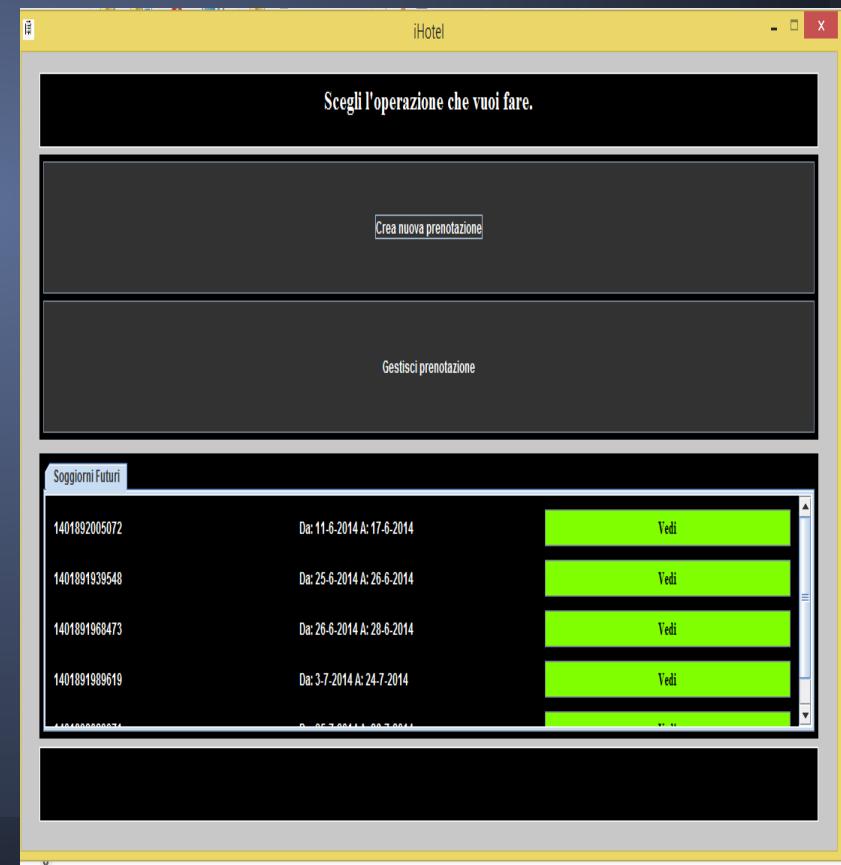
# Soluzioni ai problemi - 1 iterazione

## *Bassa flessibilità nelle classi del package View - 3*

Light:



Dark:



# Soluzioni ai problemi - 1 iterazione

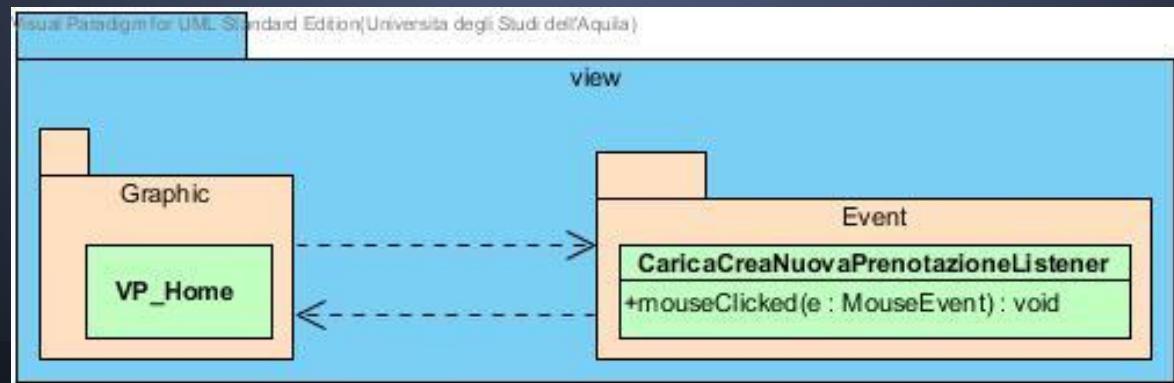
## *Separazione tra gestione eventi e struttura Frame*

Prima:

- Gestione degli eventi, assegnata alle classi già responsabili della struttura dei Frame.

Dopo:

- Classi ad-hoc per la gestione degli eventi.
- Le classi estendono dei Listener forniti da Java.
- Aumento della coesione nelle classi coinvolte.



# Soluzioni ai problemi - 1 iterazione

## *Violazione del principio “Comando-Interrogazione”*

Prima:

- Violazione del principio all'aggiunta di una nuova camera alla richiesta di soggiorno.

Dopo:

- Applicazione del Design Pattern “**Observer**”.
- La classe *Soggiorno* è il subject.
- La classe del package View è l’observer.

# Soluzioni ai problemi - 1 iterazione

## *Logica per occupazione di camere di un soggiorno - 1*

Stati della camera nel tempo: (   libera        occupata)



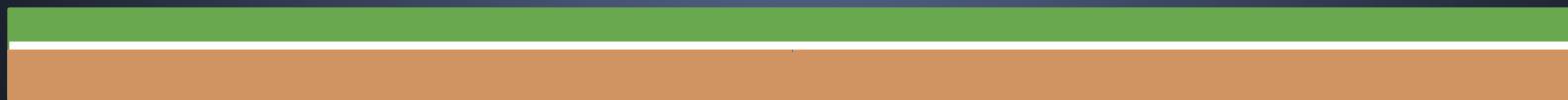
# Soluzioni ai problemi - 1 iterazione

## *Logica per occupazione di camere di un soggiorno - 2*

Il periodo in cui si effettua una richiesta di soggiorno deve ricadere in un periodo in cui la camera è libera.

Possibili situazioni: (  p. libero       p. soggiorno)

- Completa sovrapposizione tra i periodi.



- Giorno inizio soggiorno coincidente con il primo giorno periodo libero.



# Soluzioni ai problemi - 1 iterazione

## *Logica per occupazione di camere di un soggiorno - 3*

- Ultimo giorno soggiorno coincidente all' ultimo giorno del periodo libero.



- Periodo del soggiorno strettamente contenuto nel periodo libero.



Ciò richiede un algoritmo per la creazione di nuovi stati e la modifica di quelli già esistenti, in base al caso in cui ci si trova.

# Soluzioni ai problemi - 1 iterazione

## *Logica per occupazione di camere di un soggiorno - 4*

Prima:

- Utilizzo di un metodo composto di circa 340 SLOC, nella classe *Camera*.

Dopo:

- Applicazione dei Design Pattern “**State**”, “**Simple Factory**” e “**Strategy**”.
- La classe *Camera* diventa *CameraContext*.
- Al posto della classe *StatoCamera*, si introduce una gerarchia radicata nella classe astratta *CameraState*.
- Dalla gerarchia discendono le classi *CameraStateLibera* e *CameraStateOccupata*, dal comportamento differente.

# Soluzioni ai problemi - 1 iterazione

## *Logica per occupazione di camere di un soggiorno - 5*

- Attraverso il pattern Simple Factory, viene decisa la strategia concreta da utilizzare.
- Progettazione di una gerarchia di strategie, radicata nella classe interfaccia *CreaStatiCameraStrategy*.
- Progettazione di 4 classi che implementano l'interfaccia, in modo opportuno.

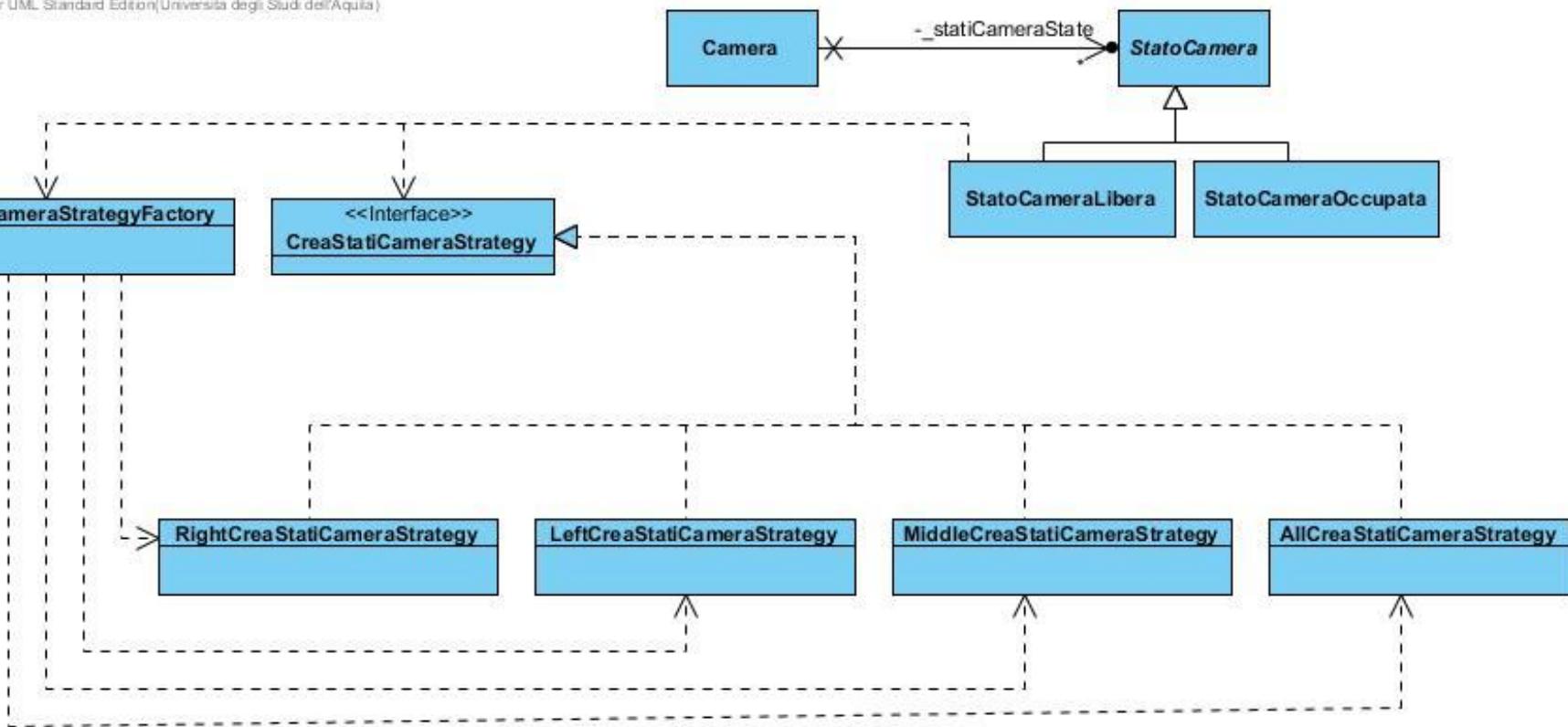
### Conseguenze:

- Aumento del numero di classi coinvolte nell'operazione.
- Alta coesione in tutte le classi coinvolte.
- Attraverso la collaborazione tra molte classi, si è ottenuta una riduzione delle SLOC a 10, nel metodo iniziale.

# Soluzioni ai problemi - 1 iterazione

*Logica per occupazione di camere di un soggiorno - 6*

Visual Paradigm for UML Standard Edition (Università degli Studi dell'Aquila)



# Analisi del dominio

- E' stata effettuata un'analisi approfondita sul concetto di servizio, all'interno di una struttura ricettiva.
- Si è osservata una differenza sostanziale tra due tipologie di servizi:
  - Servizi offerti direttamente dalla struttura ricettiva, denominati **servizi interni**.
  - Servizi offerti mediante contratti stipulati dalla struttura ricettiva con terzi (PayTv, Telefonia, ...), denominati **servizi esterni**.

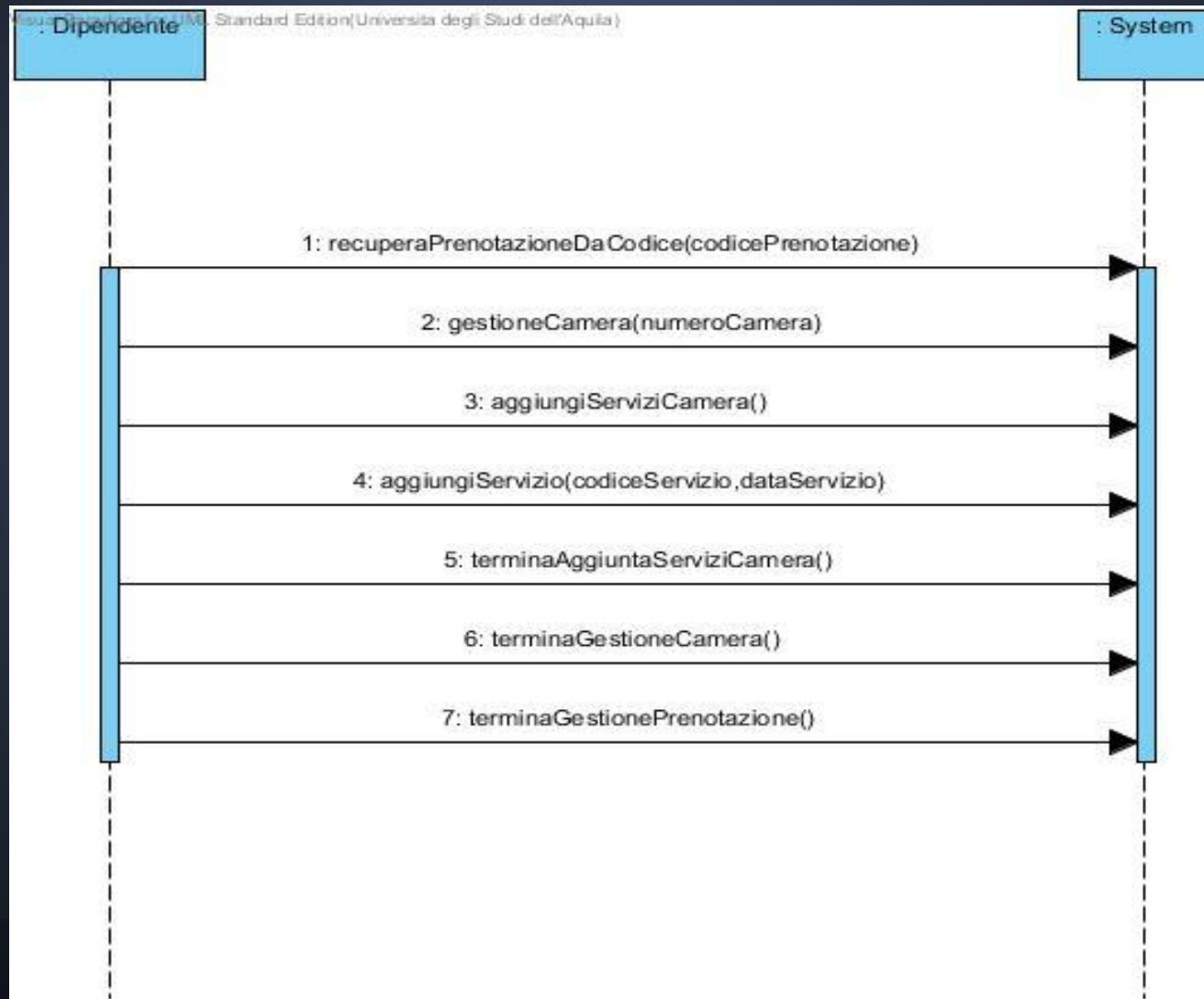
# Modifica Prenotazione

## *Aggiunta di servizio interno alla prenotazione - 1*

1. L'ospite richiede un servizio (Es. champagne in camera) per la prenotazione a suo carico, per una data.
2. L'ospite fornisce il codice della prenotazione.
3. Il dipendente richiede al sistema di recuperare la sua prenotazione, attraverso il codice.
4. Il sistema mostra i dettagli della prenotazione al dipendente.
5. L'ospite comunica in quale camera vuole usufruire del servizio.
6. Il dipendente seleziona la camera indicata dall'ospite.
7. Il dipendente sceglie il servizio da aggiungere.
8. Il dipendente inserisce la data nella quale fornire il servizio.
9. Il dipendente aggiunge il servizio alla camera.
10. Il sistema mostra l'elenco dei servizi aggiornato.
11. Il dipendente termina la gestione della camera.
12. Il dipendente termina la gestione della prenotazione.

# Modifica Prenotazione

## *Aggiunta di servizio interno alla prenotazione - 2*



# Sommario

- Processo di sviluppo
- Ideazione
- Elaborazione
  - Iterazione 1
  - Iterazione 2
  - Iterazione 3
- Conclusioni

# Obiettivi

- Modifica della struttura delle classi del package View, per ottenere un Frame unico per l'applicazione.
- Analisi delle modalità di pagamento per un soggiorno.
- Analisi del ciclo di vita di un soggiorno in una struttura alberghiera.
- Introduzione di politiche per il calcolo della caparra e del giorno limite di invio di garanzia per un soggiorno.
- Sviluppo del caso d'uso per effettuare check in.
- Sviluppo del caso d'uso per effettuare check out.

# Modifica struttura package View

Prima:

- Alla transizione da una schermata ad un'altra si chiude un Frame e se ne riapre un altro.
- Latenza percettibile nella transizione tra le schermate.

Dopo:

- Introduzione di un'unica classe di tipo Frame. denominata *ViewFrameApplication*.
- Introduzione del gestore del layout **CardLayout** per il contentPane di *ViewFrameApplication*.
- Tutti i vecchi Frame sono diventati Panel.
- Velocizzazione delle transizioni tra schermate.

# Modalità di pagamento

Sono stati analizzati diversi metodi di pagamento:

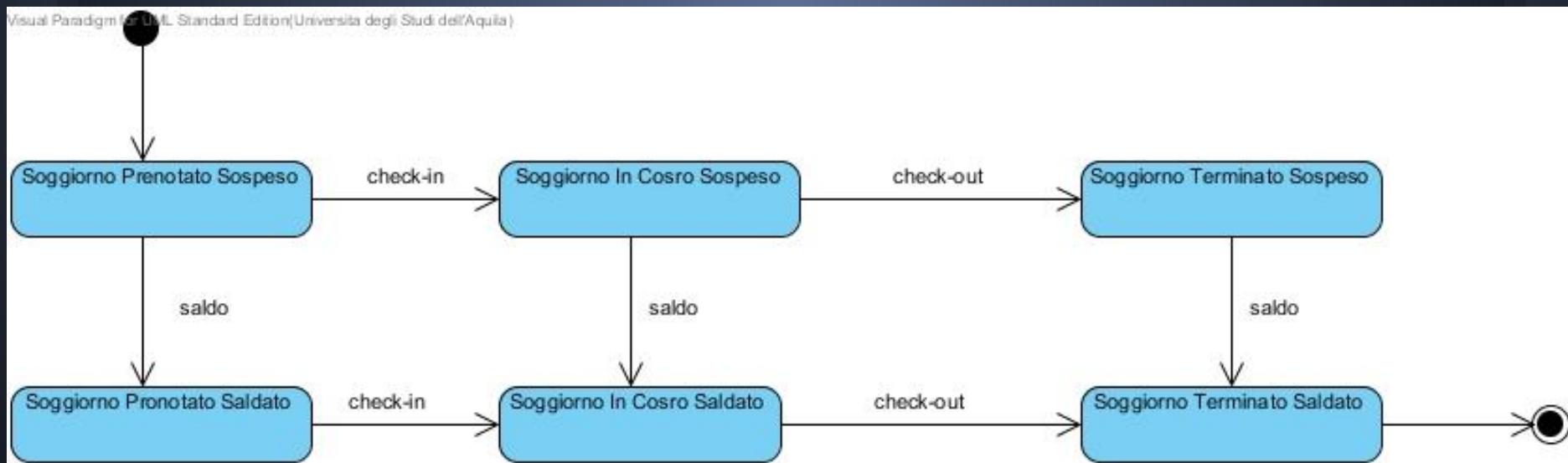
- Contanti.
- Bonifico.
- Carta di credito / Bancomat.

Per registrare un pagamento con carta è necessario avvalersi di diversi sistemi esterni:

- Lettore di carte.
- Sistema di autorizzazione al pagamento.
- Sistema di pagamento con carta.

# Analisi ciclo di vita soggiorno - 1

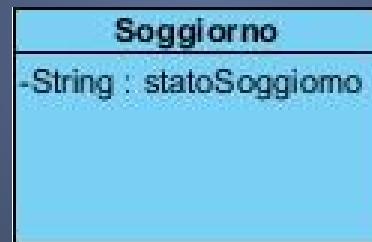
- Mediante l'aiuto di un esperto del dominio, è stata effettuata l'analisi dei possibili stati in cui si può trovare un soggiorno, in una struttura ricettiva, seguendo un flusso lineare, senza cancellazioni.



# Analisi ciclo di vita soggiorno - 2

Soluzione di base:

- Aggiungere un attributo nella classe soggiorno, indicante il suo stato.



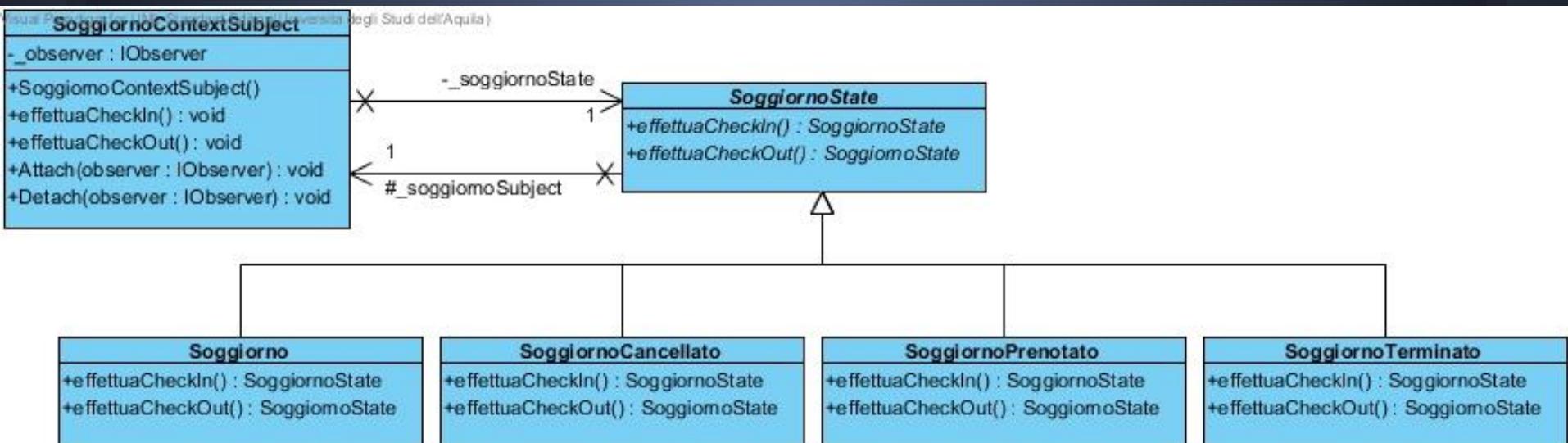
Problemi:

- Scarsa flessibilità.
- Alta frequenza di strutture condizionali per individuare in quale stato si trova il soggiorno.

# Analisi ciclo di vita soggiorno - 3

Soluzione intermedia:

- Pattern **State**.
- Aumento flessibilità.
- Minor numero di strutture condizionali per individuare in quale stato si trova il soggiorno.



# Analisi ciclo di vita soggiorno - 4

Soluzione avanzata:

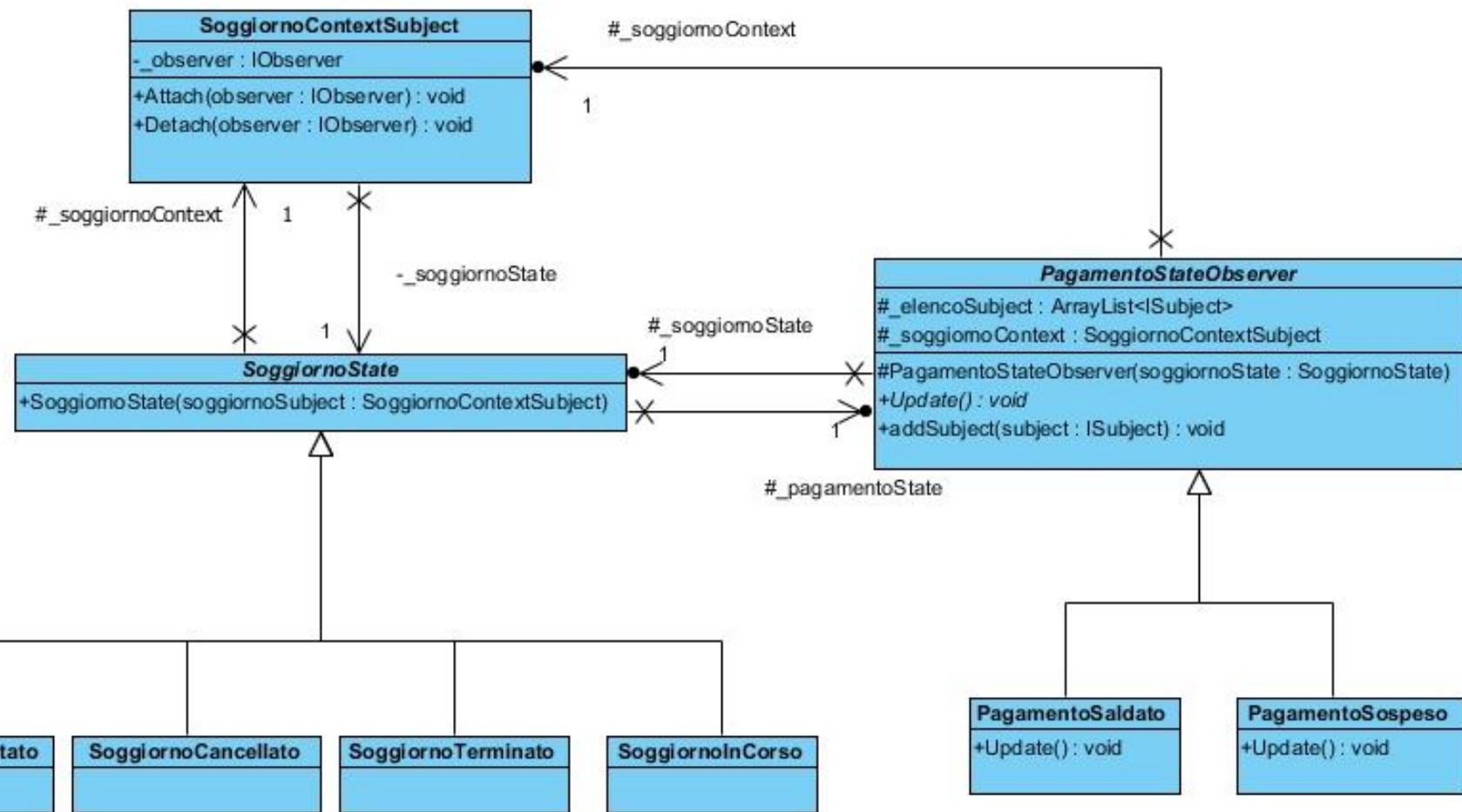
- Pattern State e Observer;
- Rimozione di strutture condizionali per capire se il soggiorno è stato saldato o meno;
- Introduzione del pattern Observer necessario al monitoraggio delle camere riservate per il soggiorno.

Esempio:

- Aggiunta di un servizio interno ad una camera del soggiorno.
- Saldo dell'ammontare del soggiorno.

# Analisi ciclo di vita soggiorno - 5

Visual Paradigm for UML Standard Edition(Università degli Studi dell'Aquila)



# Analisi ciclo di vita soggiorno - 6

Soluzione di base:



Soluzione intermedia:



Soluzione avanzata:



# Introduzione politiche aziendali - 1

- Durante il colloquio con l'esperto del dominio, si è evidenziata un'alta variabilità decisionale in merito a due temi relativi al soggiorno:
  - Calcolo dell'*ammontare della caparra* da far pagare al cliente prenotante.
  - Calcolo del *giorno di scadenza* entro il quale il cliente deve fornire una garanzia alla struttura alberghiera per far sì che essa risulti valida.
- Per entrambe sono state evidenziate diverse regole di decisione.

# Introduzione politiche aziendali - 2

Regole per il **calcolo del giorno di scadenza**:

- “Storia” del cliente prenotante.
- Occupazione delle camere, al momento della prenotazione, per il periodo richiesto.
- Lunghezza del periodo relativo al soggiorno.
- ...

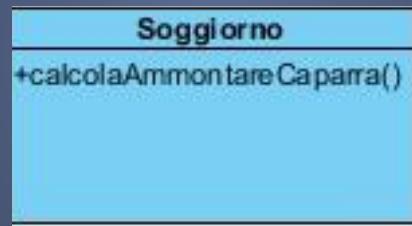
Regole per il **calcolo dell’ammontare della caparra**:

- “Storia” del cliente prenotante.
- Lunghezza del periodo relativo al soggiorno.
- ...

# Introduzione politiche aziendali - 3

Soluzione di base:

- sviluppo di un algoritmo per il calcolo della caparra, interno al soggiorno.



Problemi:

- Elevato numero di strutture condizionali.
- Struttura rigida in **contrapposizione** con politiche aziendali altamente variabili.
- Difficoltà di progettazione, lettura e manutenzione del codice.

# Introduzione politiche aziendali - 4

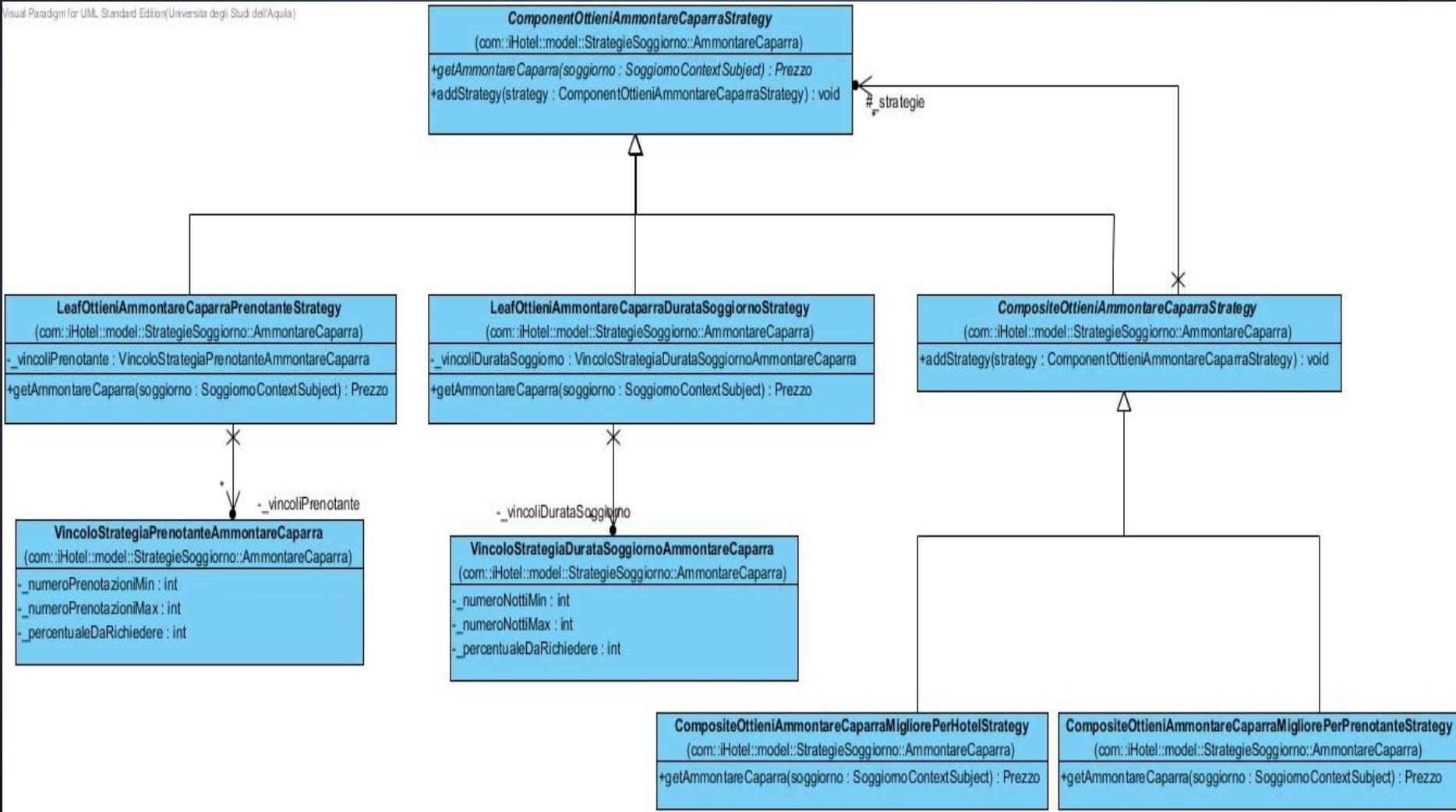
Soluzione avanzata:

- Pattern **Strategy**, **Composite** e **Simple Factory**.
- Progettazione di una classe sw per ogni algoritmo.
- Progettazione di strategie composite, per la risoluzione dei conflitti tra strategie concorrenti.
- Introduzione di una classe che instanzia le strategie.



# Introduzione politiche aziendali - 5

Visual Paradigm for UML Standard Edition (Università degli Studi dell'Aquila)



# Caso d'uso - Check in

## *Analisi e sviluppo - 1*

1. Il cliente arriva alla struttura ricettiva con una prenotazione relativa ad un soggiorno, effettuata in precedenza.
2. Il dipendente attiva il caso d'uso "Gestione soggiorno", per recuperare il soggiorno attraverso il suo codice.
3. Il sistema mostra al dipendente i dettagli del soggiorno.
4. Il dipendente comunica al cliente le condizioni del soggiorno.
5. L'ospite della soggiorno fornisce il documento.
6. Il dipendente inserisce le informazioni del documento.

*Il dipendente ripete i passi 5-6 fino al termine degli ospiti del soggiorno.*

7. Il cliente vuole versare un acconto in contanti per il soggiorno.
8. Il dipendente attiva il caso d'uso “Gestione Pagamenti” per versare un acconto.
9. Il dipendente indica al sistema di effettuare il check in degli ospiti.
10. Il sistema invia le informazioni sui clienti al sistema per la emissione delle schede di pubblica sicurezza.

# Caso d'uso - Check in

## Analisi e sviluppo - 2



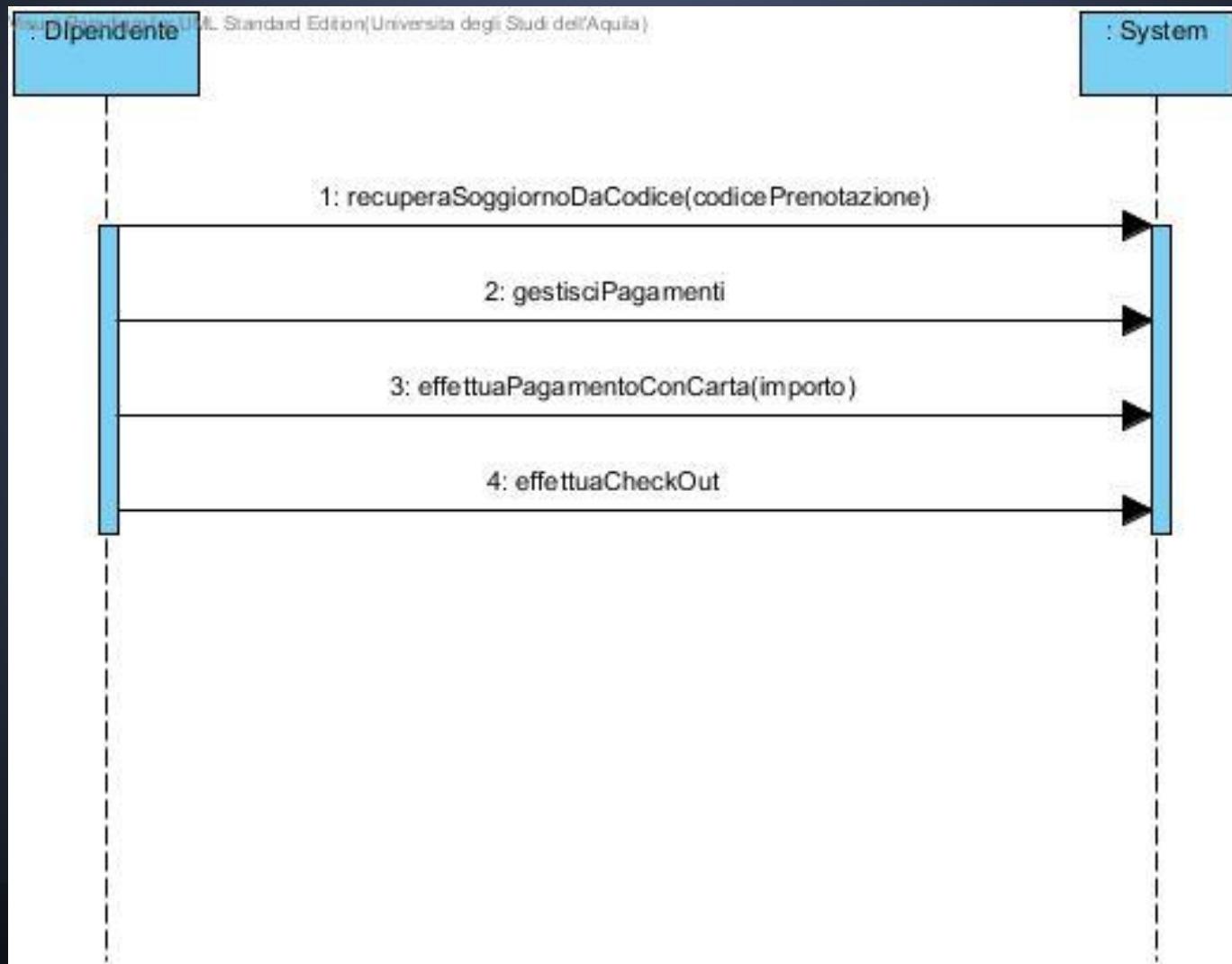
# Caso d'uso - Check out

## *Analisi e sviluppo - 1*

1. Il cliente si reca in portineria, a fine soggiorno.
2. Il cliente comunica il codice del soggiorno al dipendente.
3. Il dipendente chiede al sistema di recuperare il soggiorno comunicato dal cliente.
4. Il sistema mostra i dettagli della soggiorno.
5. Il dipendente comunica al cliente il prezzo totale del soggiorno.
6. Il cliente vuole effettuare il pagamento attraverso carta di credito / bancomat.
7. Il dipendente attiva il caso d'uso “Gestione pagamento” per effettuare il pagamento richiesto dal cliente.
8. Il dipendente comunica l'esito del pagamento al cliente.
9. Il cliente abbandona la struttura ricettiva.

# Caso d'uso - Check out

Analisi e sviluppo - 2



# Sommario

- Processo di sviluppo
- Ideazione
- Elaborazione
  - Iterazione 1
  - Iterazione 2
  - Iterazione 3
- Conclusioni

# Interfaccia grafica

## Elaborazione - Home - Iterazione 1

iHotel - Crea nuova prenotazione - Step 1 di 2

Data di inizio:

giugno						
dom	lun	mar	mer	gio	ven	sab
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5
6	7	8	9	10	11	12

2014 ▲ ▼ ►

Data di fine:

giugno						
dom	lun	mar	mer	gio	ven	sab
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5
6	7	8	9	10	11	12

2014 ▲ ▼ ►

Today: 6-giu-2014 Clear Today: 6-giu-2014 Clear

Tipologie:

Singola

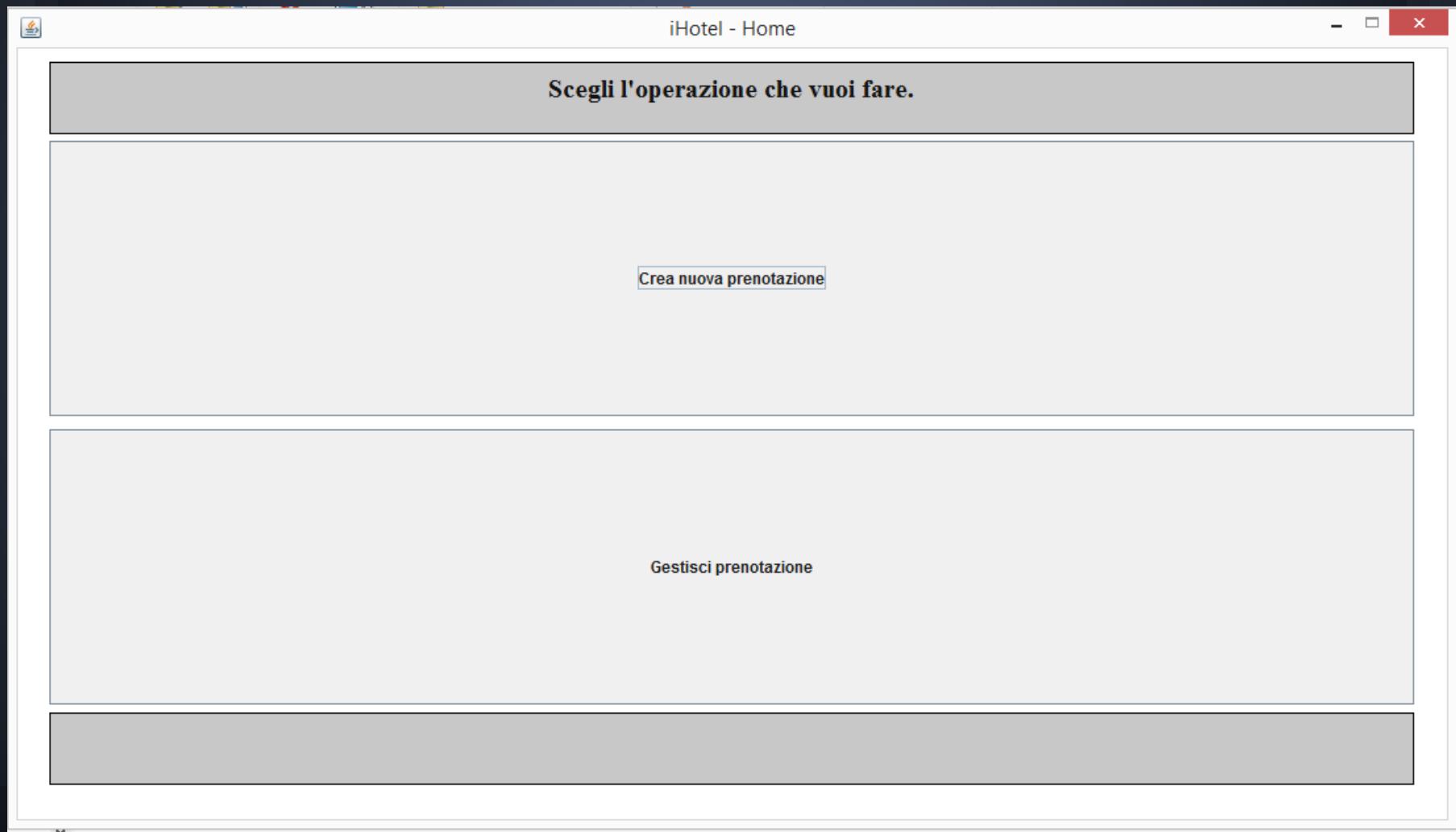
Tripla

Doppia

**Avanti**

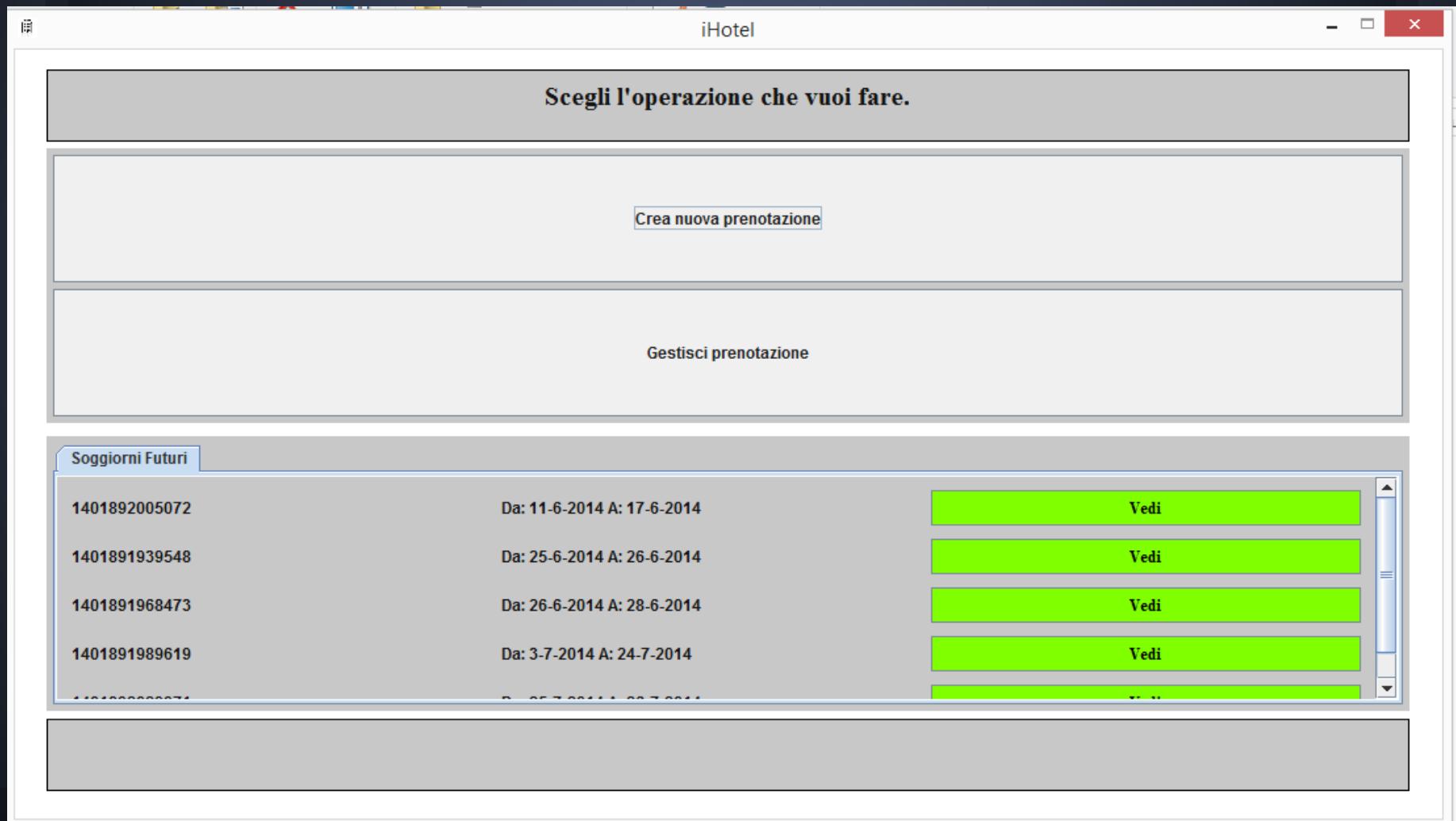
# Interfaccia grafica

## Elaborazione - Home - Iterazione 2



# Interfaccia grafica

## Elaborazione - Home - Iterazione 3



# Codice prodotto

## Elaborazione - Iterazione 1

Source Files: 20

Directories: 9

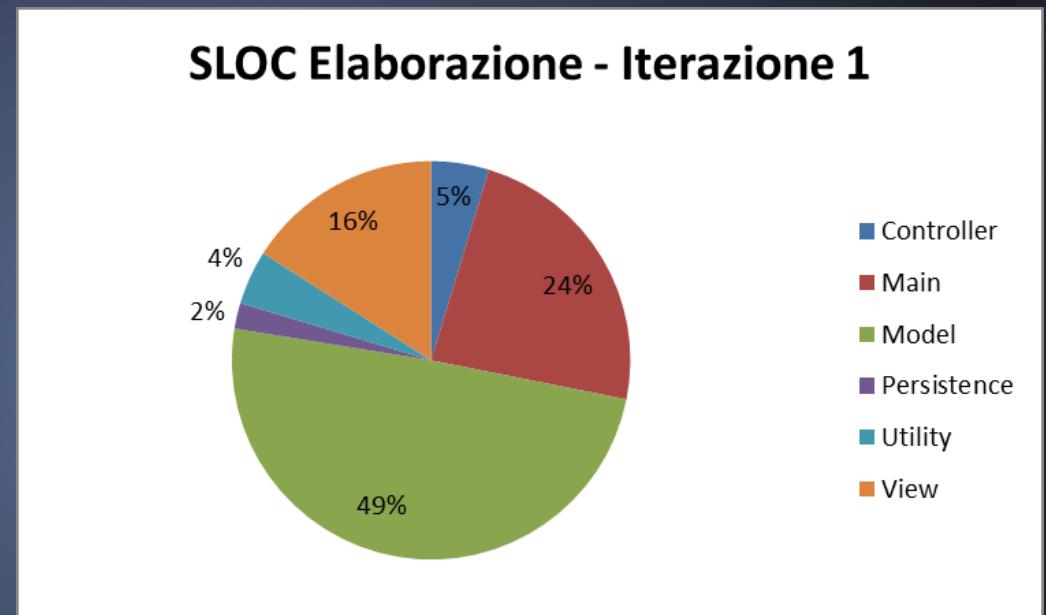
LOC: 3081

BLOC: 317

SLOC-L: 1370

SLOC-P: 1635

CLOC: 1129



# Codice prodotto

## Elaborazione - Iterazione 2

Source Files: 72

Directories: 26

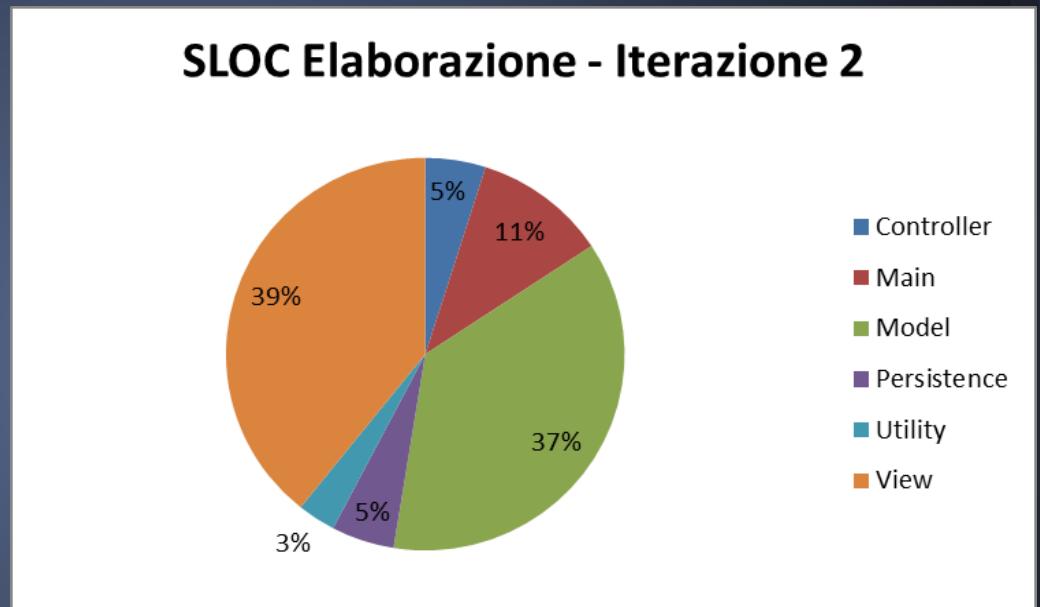
LOC: 7168

BLOC: 660

SLOC-L: 2928

SLOC-P: 3640

CLOC: 2868



# Codice prodotto

## Elaborazione - Iterazione 3

Source Files: 152

Directories: 48

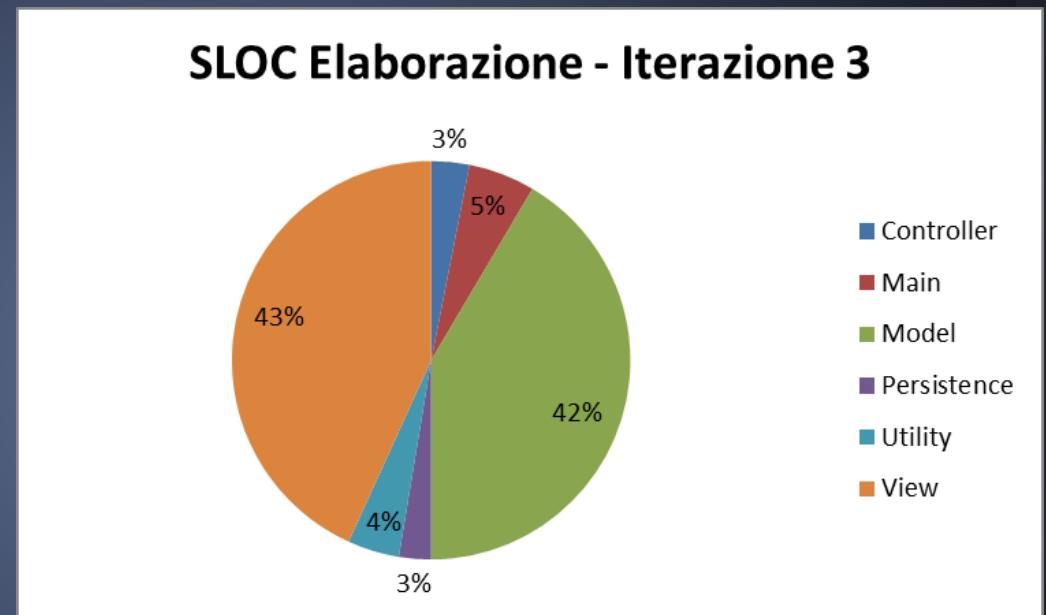
LOC: 15316

BLOC: 1655

SLOC-L: 5675

SLOC-P: 7192

CLOC: 6469



# Casi d'uso affrontati

Sono stati sviluppati i seguenti casi d'uso:

- Creazione della richiesta di soggiorno.
- Aggiunta di servizi interni ad un soggiorno.
- Check in.
- Check out.

Mediante le scelte effettuate, attraverso il processo di sviluppo utilizzato, è stato sviluppato il “**core**” del sistema, ruotante attorno al concetto di soggiorno.

Fine