

Progetto "Residence Stella"

Relazione Progetto Prog. ad Oggetti di Nicola Lazzarin
2020/2021 - Nicola Lazzarin - Irene Benetazzo

INTRODUZIONE

Residence Stella è un progetto per la visualizzazione di strutture appartenenti ad un residence. Il tipo di target per questa applicazione sono i clienti, i quali possono interagire con l'applicazione per visualizzare i vari tipi di strutture che il residence mette a disposizione.

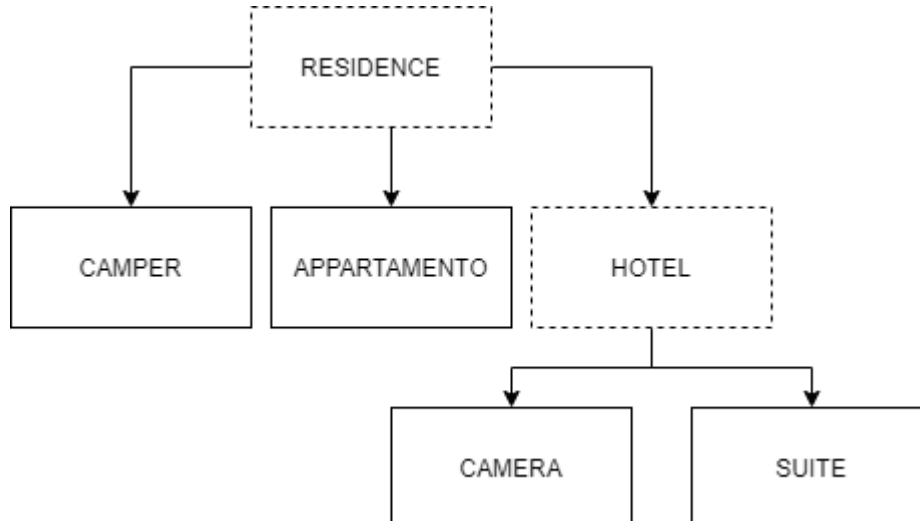
i vari dati che vengono visualizzati sono:

- La tipologia di struttura scelta
- Il costo a notte con annessi servizi selezionati
- La disponibilità delle strutture e i vari servizi che mette a disposizione

Il design-pattern utilizzato è Model-View-Controller, abbiamo cercato di suddividere al meglio la parte della GUI con quella del modello cosicché in un futuro si possa riusare per altri scopi.

GERARCHIA

La gerarchia rappresenta tutte le strutture e le loro derivate, gli oggetti che potranno essere istanziati hanno il bordo con linea continua, invece le classi astratte (non istanziabili) hanno il bordo con linea tratteggiata.



La classe base astratta polimorfa è Residence, permette la memorizzazione di dati comuni come la dimensione della struttura (in Mq), il numero dei letti e lo stato in cui si trova, se libero, occupato o lavori.

La classe Camper eredita direttamente da Residence aggiungendo e memorizzando i vari servizi che una piazzola per camper può avere a disposizione.

La classe Appartamento eredita direttamente da Residence aggiungendo e memorizzando anch'essa i vari servizi base che un appartamento può avere a disposizione.

La classe Hotel eredita dalla classe Residence e implementa i servizi base che un Hotel può offrire ai propri clienti, come ad esempio la piscina e il pacchetto mezza pensione.

Da Hotel ereditano le classi Camera e Suite, Camera fa un semplice calcolo dei costi complessivi, invece Suite aggiunge e memorizza anche altri servizi inerenti ad una suite.

TEMPLATE

deep_unique_ptr.h è una classe templetizzata che implementa la **gestione profonda** della memoria degli oggetti.

vettore.h è una classe templetizzata “contenitore” simile a quella STL ma più breve. La classe implementa tutte le funzioni basilari necessarie, come gli iteratori, per un corretto funzionamento di essa.

CHIAMATE POLIMORFE

Nella gerarchia sono presenti questi metodi virtuali

- Residence* clone() const=0 (gestione memoria profonda con deep_unique_ptr)
- double costoPiscina()=0 (calcola il costo della piscina)
- double costoSpiaggia()=0 (calcola il costo della spiaggia)
- double prezzo()=0 (calcola il prezzo totale di una notte con i servizi e le persone)
- aumentoStagione(periodo p) (aumenta in percentuale il prezzo in base al periodo)
- double costoRistorante() (calcola il costo del ristorante)
- puliziaFinale() (calcola il costo della pulizia, solo appartamento)
- costoAcqua() (calcola il costo dell'acqua per persona, solo camper)
- costoCorrente() (calcola il costo della corrente per persona, solo camper)
- clone() (clona gli oggetti e rende possibile la gestione della memoria)
- vari distruttori virtuali

DESCRIZIONE FUNZIONAMENTO GUI

Per l'interfaccia grafica abbiamo deciso di usare 4+1 (popup dialog) schermate.

Tutte e 4 le schermate sono inizializzate dalla classe View, la quale crea e tiene i puntatori alle 4 finestre, il popup dialog invece è gestito dalla classe StructureView.

Per la gestione delle schermate abbiamo utilizzato la classe Controller con la tecnologia che qt fornisce dei *signals* e *slots*.

Lo scambio delle interfacce avviene tramite la classe View con la funzione setCurrentWidget, la classe View eredita QStackedWidget in modo tale da visualizzare l'interfaccia con i propri widget solo una alla volta.

Per la classe **Home** (interfaccia principale) si è deciso di ridefinire la classe Label per aggiungere la funzionalità del lancio di un *signal* quando l'immagine viene premuta dal puntatore del mouse.

Grazie a questa ridefinizione si può collegare il QRadioButton alle immagini associate in modo da rendere più intuitibile l'usabilità del programma. I QRadioButton sono stati scelti perchè grazie a loro si forza la scelta di solo una opzione. Per il numero di Ospiti si è usata la classe QSpinBox con limite degli ospiti a 8 (per struttura).



BENVENUTI AL RESIDENCE STELLA



NUMERO OSPITI
(per struttura)

Seleziona il periodo

hitbox →

| | | |
|----------------------------------|-----------------------|-----------------------|
| <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> |
| € | €€ | €€€ |
| Bassa Stagione | Media Stagione | Alta Stagione |

Seleziona la Struttura

| | | | |
|----------------------------------|-----------------------|-----------------------|-----------------------|
| <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |
| | | | |
| Camera | Suite | Appartamento | Piazzola Camper |

PROSEGUI

La classe dei **Servizi** è dipendente dalla scelta presa nella classe home, questo è stato deciso così da non “affollare” troppo il programma di interfacce/view inutili.

La view servizi cambia “aspetto” in base a quale struttura si seleziona nella schermata home grazie alla funzione setVisible dei widget, in questo modo quando si seleziona ad esempio Camera si vedranno solamente i pulsanti dei servizi che la camera mette a disposizione.

Come tipologia di pulsanti si è deciso di usare QRadioButton per la selezione forzata del Ristorante e QCheckBox per gli altri servizi.

Dopo aver selezionato la struttura e i servizi, cliccando sul pulsante prosegui, prima viene richiesto automaticamente il popolamento del vettore con la classe database poi viene caricata la schermata di visualizzazione della tabella con tutte le opzioni disponibili.

Il vettore viene caricato solo con la struttura selezionata in modo da evitare caricamenti inutili però quando si ritorna alla schermata home, tramite il pulsante home nella tabella o back in servizi, il vettore viene eliminato e reso disponibile per un nuovo caricamento.

Nella tabella, come nella view servizi, vengono caricati i dati in base alla struttura selezionata assieme alle righe e colonne esatte.

| | |
|---|------------------------------|
|  | Struttura adatta ai disabili |
|  | Struttura disponibile |
|  | Struttura già prenotata |
|  | Struttura in manutenzione |
|  | Lavatrice |
|  | N°ospiti superiore ai Letti |








In alto a destra si trova un pulsante con il punto di domanda (?), quel pulsante serve per fare il popup dell'info dialog che spiega tutta la simbologia della tabella. Il popup viene gestito dalla classe tabella StructureView.

Se si vuole approfondire che servizi, prezzo e altre informazioni utili si fa il doppio click sulla riga con la struttura d'interesse e comparirà la view di informazione della struttura.

Come con le altre schermate anche la schermata delle informazioni cambia in base a quale struttura si è selezionata all'inizio. Per il popolamento dell'interfaccia informazioni si è deciso di usare la classe vettore di QLabel*. Questo vettore viene "riempito", in base al riconoscimento degli oggetti nel vettore principale, di QLabel inerenti alla struttura e poi aggiunto in ordine nel MainLayout.

ESEMPIO SCHERMATA ELENCO STRUTTURE

★ Residence — □ ×

| Elenco Strutture | | | | | |
|------------------|---|---------|-------|---|------------|
| Struttura | Stato | N°Letti | Piano | Disabile | Prezzo Min |
| 1 |  | 2 | 0 |  | 30 |
| 2 |  | 2 | 3 | | 35 |
| 3 |  | 2 | 5 | | 35 |
| 4 |  | 2 | 4 | | 35 |
| 5 |  | 2 | 0 |  | 40 |

↑
doppio click

ESEMPIO SCHERMATA SERVIZI E INFORMAZIONI

Informazioni e Servizi Struttura

| | |
|--|---|
| Dimensione in Mq: 12 | Stato Struttura: Prenotabile |
| Numero Letti: 2 | Per 1 Ospite nel periodo di Bassa Stagione Il Prezzo a notte è di: 40.5€ |
| La camera si trova al 3° Piano | Per la prenotazione o ulteriori informazioni si prega di telefonare al residence |
| Opzione di alloggio: Solo colazione | |
| Servizio Incluso: Accesso Piscina Servizi aggiuntivi selezionati: Nessun servizio aggiuntivo selezionato | |

BACK

CARICAMENTO RISORSE

Per il caricamento delle risorse (immagini e database) abbiamo usato il file .QRC (Qt Resource System) il quale ci permette di inserire le risorse direttamente nell'eseguibile.

CARICAMENTO STRUTTURE (INPUT)

Per il caricamento delle strutture si è deciso di usare un file di testo .txt che viene letto dalla classe database usando QFile e QTextStream. La classe, grazie all'alias del modello, costruisce subito gli oggetti e li aggiunge mano a mano che legge i vari input da file.

Per la "conversione" da stringa a oggetto si è deciso di usare un metodo nostro di lettura dei dati.

Nel file .txt è spiegato tutto il necessario per aggiungere nuove strutture e altro.

Nella consegna **sono inclusi i file input di esempio** (database.txt)

SUDDIVISIONE DEL LAVORO

Il progetto è stato sviluppato assieme a Irene Benetazzo la quale ha collaborato con la decisione dei vari criteri di implementazioni delle classi nel modello e nella GUI. Successivamente abbiamo deciso di suddividere il lavoro in questo modo: Nicola Lazzarin si è occupato principalmente della GUI e del file di input, mentre Irene Benetazzo, oltre a consigliarmi la disposizione delle varie view si è occupata principalmente del modello, vettore e unique ptr.

| Irene Benetazzo | | Nicola Lazzarin | |
|-------------------|------------------|------------------------|--------------------------|
| HEADERS | CPP'S | HEADERS | CPP'S |
| vettore.h | | clickablelabel.h | clickablelabel.cpp |
| deep_unique_ptr.h | | home.h | home.cpp |
| residence.h | residence.cpp | infodialog.h | infodialog.cpp |
| hotel.h | hotel.cpp | servizi.h | servizi.cpp |
| camera.h | camera.cpp | structuredetailsview.h | structuredetailsview.cpp |
| suite.h | suite.cpp | structureview.h | structureview.cpp |
| appartamento.h | appartamento.cpp | controller.h | controller.cpp |
| camper.h | camper.cpp | view.h | view.cpp |
| model.h | model.cpp | | database.cpp |
| | | | main.cpp |

ORE PERSONALI IMPIEGATE

| | |
|---|-----|
| Analisi del problema, progettazione dell'intero progetto e suddivisione dei compiti | 5h |
| Progettazione GUI | 3h |
| Apprendimento QT | 14h |
| Programmazione | 25h |
| Grafica / Abbellimento | 3h |
| Test e Debugging | 4h |
| Scrittura relazione | 2h |
| TOTALE | 55h |

Ho sforato di 5 ore perché, oltre al Debugging, volevo “abbellire” di più il progetto con grafiche personalizzate e loghi.

COMPILAZIONE

Per la compilazione è sufficiente eseguire:

```
mkdir build
```

```
cd build
```

```
qmake ../Residence.pro
```

```
make
```

```
./Residence
```

NOTE

Il progetto è stato sviluppato e testato su macchina windows 10 pro 64 bit con QT 5.15.2 e compilatore MinGW 8.1.0 64 bit, ulteriori test di verifica sono stati fatti sulla macchina virtuale Ubuntu fornita dal professore, con versione qt 5.9.5 e g++ 5.4.0 con esito positivo, in altri sistemi potrebbe non funzionare come previsto.

Per il funzionamento del programma c'è bisogno del file di input, senza di esso il programma non parte.