Sommario

[Traccia Progetto: 2](#_Toc158654987)

[Spiegazione Progetto 4](#_Toc158654988)

[LOGIN 4](#_Toc158654989)

[Autenticazione Amministratore 4](#_Toc158654990)

[Autenticazione Cliente 6](#_Toc158654991)

[6](#_Toc158654992)

[Singleton Pattern: 7](#_Toc158654993)

[Singleton Pattern Utilizzato: 7](#_Toc158654994)

[7](#_Toc158654995)

[INTERFACCIA ADMIN 8](#_Toc158654996)

[Schermata Home Amministratore 8](#_Toc158654997)

[Bottone 1 - Aggiungi Bici a Parcheggio 9](#_Toc158654998)

[Pattern Builder: 10](#_Toc158654999)

[Pattern Builder Utilizzato: 11](#_Toc158655000)

[Bottone 2 – Aggiungi Equipaggiamento 12](#_Toc158655001)

[Bottone 3 – Aggiungi / Aggiorna Tariffe 13](#_Toc158655002)

[Bottone 4 – Percentuale Utilizzo 14](#_Toc158655003)

[INTERFACCIA CLIENTE 15](#_Toc158655004)

[Schermata Home Cliente 15](#_Toc158655005)

[Bottone 1 – Prenota Bici 16](#_Toc158655006)

[Pattern Mediator: 18](#_Toc158655007)

[Pattern Mediator Utilizzato: 19](#_Toc158655008)

[Bottone 2 – Restituisci Bicicletta 20](#_Toc158655009)

[Bottone 3 – Pagamento 22](#_Toc158655010)

[Pattern Factory Method: 23](#_Toc158655011)

[Pattern Factory Method Utilizzato: 24](#_Toc158655012)

[DATABASE: 25](#_Toc158655013)

[Relazionale 25](#_Toc158655014)

# Traccia Progetto:

**# progetto\_prog3**

**# Traccia - Biciclette**

**Si vuole sviluppare un’applicazione per la gestione di Biciclette in affitto. Si suppone di**

**avere una flotta di biciclette posizionate in appositi parcheggi di una città. Ogni**

**parcheggio può contenere n biciclette appartenenti a k categorie (i.e. passeggio,**

**corsa, ...) e ogni bicicletta può avere il suo equipaggiamento (i.e., casco, guantini, . . .).**

**Si vuole gestire il sistema in modalità amministratore e cliente.**

**## Amministratore**

**# L’amministratore può:**

**aggiungere una nuova bicicletta**

**aggiungere nuovi equipaggiamenti**

**aggiornare le tariffe orarie per l’affitto**

**visualizzare le percentuali di utilizzo per ogni bicicletta e per ogni categoria**

**## Cliente**

**# Il cliente può:**

**prenotare una bicicletta (via sms o e-mail) e ritirarla, insieme**

**all’equipaggiamento, presso un parcheggio;**

**restituzione della bicicletta che può avvenire anche in un parcheggio diverso da**

**quello del ritiro;**

**pagamento della corsa secondo le modalità: contanti, carta di credito o bancomat. Il**

**costo della corsa viene stabilito automaticamente in base ai chilometri effettuati;**

**Sviluppare un programma per la simulazione del sistema, garantendo le opportune**

**interfacce grafiche.**

**## Requisiti**

**La prova d’esame richiede la progettazione e lo sviluppo della traccia proposta. Lo**

**studente può scegliere di sviluppare il progetto nelle due modalità: Applicazione**

**Web o programma standalone con supporto grafico.**

**Il progetto deve essere sviluppato secondo le seguenti linee:**

**usare almeno due pattern per persona (almeno uno per chi sceglie la modalità**

**Web Application) tra i design pattern noti;**

**attenersi ai principi della programmazione SOLID;**

**usare il linguaggio Java;**

**inserire sufficienti commenti (anche per Javadoc) e annotazioni;**

**gestione delle eccezioni;**

**usare i file o database;**

**E' possibile costruire l'applicazione standalone con supporto grafico tramite l'utilizzo di**

**strumenti per la realizzazione di interfacce grafiche presenti in molti IDE (GUI Designer in**

**IntelliJ e WindowsBuilder in Eclipse) oppure utilizzare tools compatibili con JavaFx come**

**Scene Builder (compatibile con gli IDE).**

**## Consegna**

**Lo studente deve presentare una relazione sintetica (per chi usa latex è possibile**

**scaricare un template dalla piattaforma e-learning). La relazione deve contenere:**

**una breve descrizione dei requisiti del progetto;**

**il diagramma UML delle classi;**

**altri diagrammi se opportuni;**

**parti rilevanti del codice sviluppato;**

**Per chi usa latex si consiglia di utilizzare la piattaforma Overleaf:**

**La consegna potrà avvenire tramite email a tutti i docenti, con in allegato un archiovio**

**con tutto relazione, codice e presentazione. In alternativa è possibile utilizzare**

**Microsoft Teams con le stesse modalità.**

**## Modalità di esame**

**La prima parte della prova di esame verterà sulla discussione del progetto. Lo**

**studente deve preparare una presentazione sintetica (slide) per descrivere il**

**progetto svolto. La seconda parte della prova verterà sulla discussione degli**

**argomenti affrontati a lezione.**

# Spiegazione Progetto

## LOGIN

### Autenticazione Amministratore

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Quando si preme il bottone identificato dall’id “Amministratore” appare il pulsante “Login Admin” il quale possiede un handler che ci permette di aprire la finestra modale per eseguire il login da

amministratore

Immagine che contiene testo, schermata, software, Software multimediale

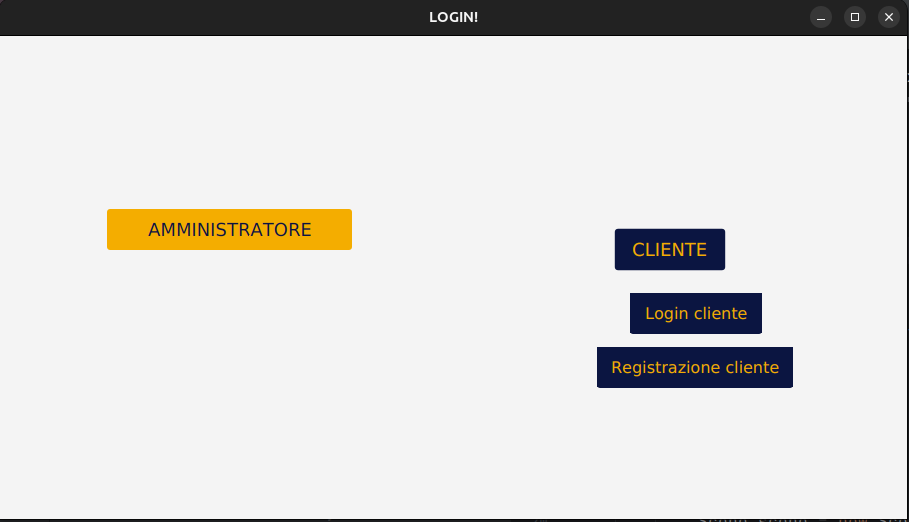
Descrizione generata automaticamente

Una volta inserite le credenziali in questa Finestra, se queste ultime sono corrette, ci appare un messaggio di dialogo di tipo “JOptionPane” il quale ci conferma di poter procedere nell’app.

In caso contrario avremmo un messaggio diverso.

Discorso quasi uguale per quanto riguarda il cliente, con l’unica differenza che quest’ultimo presenta un bottone aggiuntivo

## Autenticazione Cliente



Il bottone Registrazione cliente farà apparire a sua volta una finestra modale che ci permetterà di registrarci.

## 

Il bottone aggiuntivo è Registrazione Cliente il quale ci permette di registrare un cliente all’interno del Database.

In caso di errore comparirà un’eccezione con il relativo messaggio per comunicarci qual è stata la causa del problema

Per realizzare la Connessione al DB ho utilizzato il pattern Singleton così che la connessione a quest’ultimo potesse avvenire in un unico modo Globalmente

### Singleton Pattern:

*Assicurare che una classe abbia una sola istanza e fornire un punto*

*globale di accesso ad essa*

Immagine che contiene testo, linea, schermata, Carattere

Descrizione generata automaticamente

### Singleton Pattern Utilizzato:

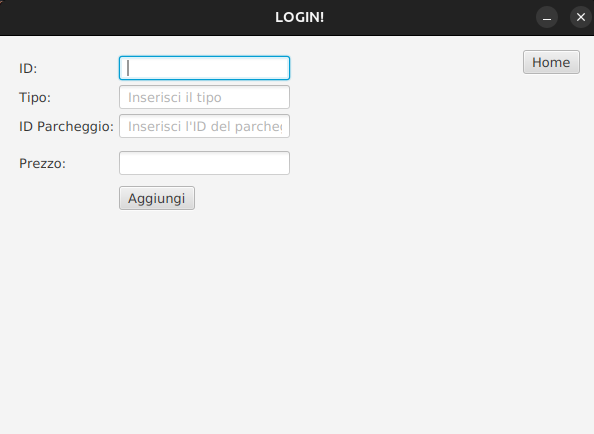
### Immagine che contiene testo, schermata, Carattere, numero Descrizione generata automaticamente

## INTERFACCIA ADMIN

### Schermata Home Amministratore

Presenta 4 bottoni con i rispettivi Nomi, ognuno dei quali gestito con un handler all’interno del controller “HelloController.java”

### Bottone 1 - Aggiungi Bici a Parcheggio



Appaiono 4 Label e 4 TextField con i rispettivi nomi Significativi e i rispettivi Placeholder gestite all’interno del Controller “PrenotazioneBicicletteController.java”

Il pulsante Aggiungi, preleva il testo contenuto nei TextField e lo manda in input ad una query di inserimento.

L’aggiunta della bicicletta avviene tramite il pattern Builder, in quanto ogni volta che l’amministratore va ad aggiungere una Bicicletta ne crea automaticamente una.

### Pattern Builder:

*Separare la costruzione di un oggetto complesso dalla sua*

*rappresentazione in modo tale che lo stesso processo di*

*costruzione può creare differenti rappresentazioni*

Immagine che contiene testo, schermata, linea, numero

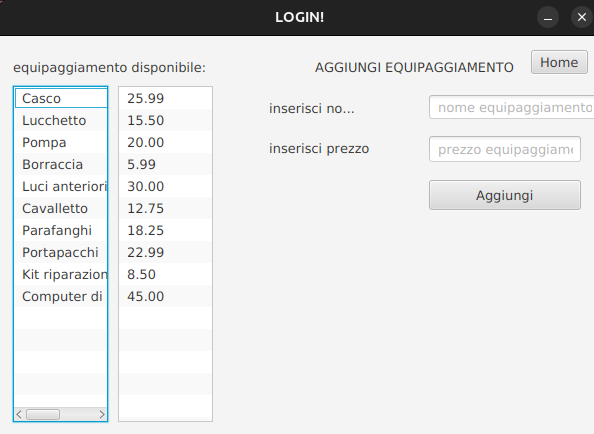
Descrizione generata automaticamente

### Pattern Builder Utilizzato:

Immagine che contiene testo, schermata, software

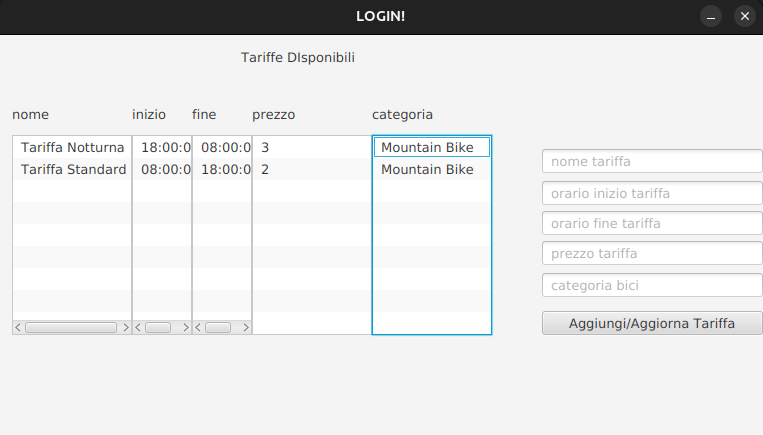
Descrizione generata automaticamente

### Bottone 2 – Aggiungi Equipaggiamento



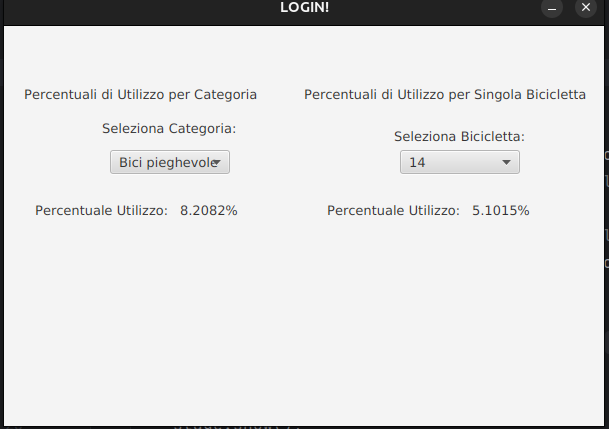
A differenza del caso Precedente oltre ad avere solamente TextField e Label abbiamo anche due listView le quali vengono precaricate (tramite la funzione initialize) alle pressione del pulsante “Aggiungi Equipaggiamento nella schermata adminHome”. E mostrano gli Equipaggiamenti presenti all’interno del Database.

### Bottone 3 – Aggiungi / Aggiorna Tariffe



In questa Scena come nella precedente ci sono le listView ciò che la rende diversa però è la funzionalità del pulsante “Aggiungi / Aggiorna Tariffe” in quanto, per ogni Tariffa inserita all’interno delle textField va a verificare se quest’ultima è già presente all’interno del DB e in caso affermativo, non la aggiunge ma ne aggiorna i campi.

### Bottone 4 – Percentuale Utilizzo



In questa scena c’è un nuovo elemento, le ChoiceBox.

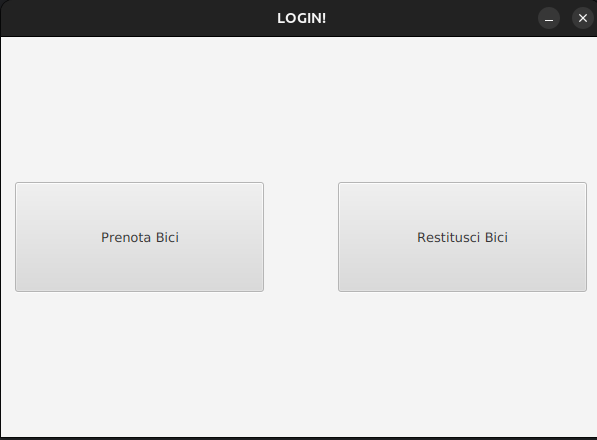
La prima è riempita con valori del Database 🡪 Biciclette.categoria\_bici

La seconda è riempita con valori del Database 🡪 Biciclette.id\_bici

Ogni volta che viene selezionato un elemento dalla tendina le rispettive Label si aggiornano, prelevando valori dal DB, tramite l’utilizzo una query.

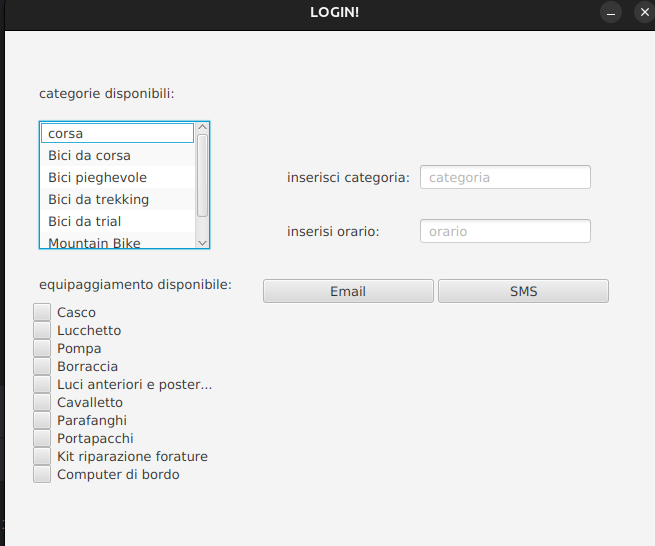
## INTERFACCIA CLIENTE

### Schermata Home Cliente



Interfaccia provvista solo di 2 Pulsanti differentemente da quanto riferito nella Traccia del Progetto in quanto la Progettazione di quest’ultimo ha richiesto che il pagamento avvenisse solamente dopo la consegna delle biciclette

### Bottone 1 – Prenota Bici



L’interfaccia ci Propone molti elementi.

In Primis analizziamo la listView, non è altro che una lista contente le categorie di tutte le biciclette che si trovano Nei Parcheggi e che quindi possono essere prenotate.  
Il requisito fondamentale di prenotazione risiede in un campo del DB e si riferisce a Bicicletta.disponibile = true.

L’equipaggiamento Disponibile è generato a Livello codice quindi per ogni elemento nella Tabella Equipaggiamento si va a creare una nuova JCheckBox di tipo Equipaggiamento, la quale verrà aggiunta alla VBox che le contiene tutte.

try {

            String query = "SELECT \* FROM dbBike.Equipaggiamento";

            ResultSet rs = db.query(query);

            while (rs.next()) {

                String categoria = rs.getString("nome\_equipaggiamento");

                CheckBox checkBox = getCheckBox(categoria);

                vBoxcheckboxContainer.getChildren().add(checkBox); // Aggiunge il checkbox al VBox

            }

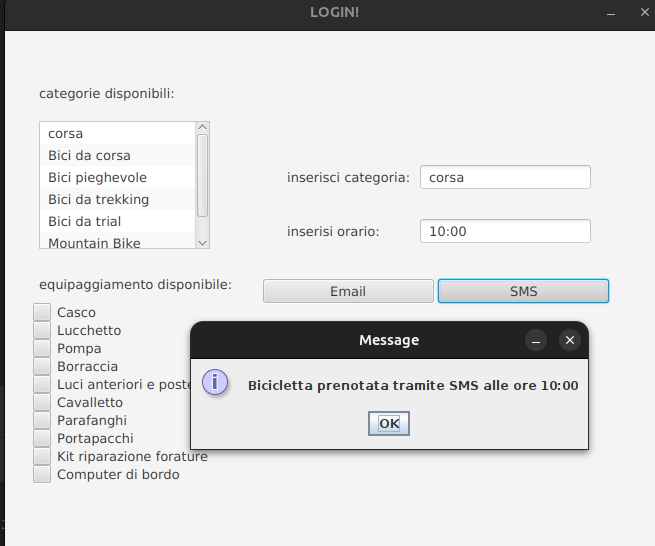
        } catch (SQLException e) {

            e.printStackTrace();

        }

La prenotazione può essere Effettuata tramite Email o SMS, implementata con l’utilizzo del pattern Mediator.

Riceviamo un messaggio di conferma della Prenotazione:



### Pattern Mediator:

*Definire un oggetto che incapsula il meccanismo di*

*interazione di oggetti, consentendo il loro*

*disaccoppiamento in modo da variare facilmente le*

*interazioni tra di loro.*

Immagine che contiene testo, diagramma, linea, Parallelo

Descrizione generata automaticamente

### Pattern Mediator Utilizzato:

Immagine che contiene testo, schermata, diagramma, Carattere

Descrizione generata automaticamente

### Bottone 2 – Restituisci Bicicletta



Questa scena segue il ragionamento delle checkbox fatto per la scena “Aggiungi Equipaggiamento”

Inoltre nella seconda listView troviamo il prezzo Stimato di ogni singola corsa effettuata con le Bici (prezzo a km) con l’eventuale sconto in caso la categoria di Biciclette rientri nelle Categorie contenute nella Tabella Tariffa.

Per ogni checkBox selezionata e deselezionata la Label contente il prezzo totale si aggiorna rispettivamente addizionando o sottraendo il prezzo della Bicicletta presa in considerazione.

Implementato con un eventListener:

 private CheckBox getCheckBox(int categoria,int prezzo) {

        CheckBox checkBox = new CheckBox(String.valueOf(categoria));

        int valore = prezzo;

        checkBox.setOnAction(event -> {

            if (checkBox.isSelected()) {

                categorieSelezionate.add(checkBox.getText());

                prezzoTotale+=valore;

                id\_bici.add(checkBox.getText());

                for (String s : id\_bici) {

                    System.out.println(s);

                }

               labelPrezzoTotale.setText(prezzoTotale+"€");

//                        System.out.println("prezzo totale:"+prezzoTotale);

            } else {

                categorieSelezionate.remove(checkBox.getText());

                prezzoTotale-=valore;

                id\_bici.remove(checkBox.getText());

               labelPrezzoTotale.setText(prezzoTotale+"€");

//                System.out.println("prezzo totale:"+prezzoTotale);

            }

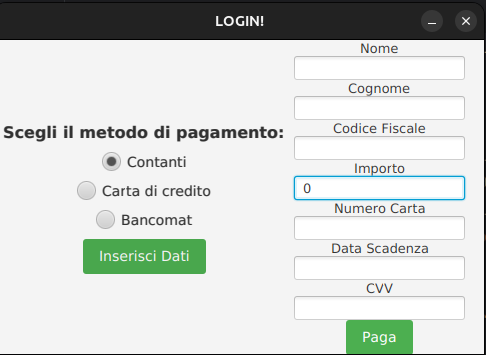
            System.out.println(categorieSelezionate);

        });

        return checkBox;

    }

### Bottone 3 – Pagamento



In questa Sezione l’utente decide la modalità di pagamento.

Quelli in foto sono tutti i campi, ma in Caso l’utente decida di voler pagare in Contanti, i Campi

“Numero Carta”, “Data Scadenza”, “CVV” vengono resi invisibili richiamando il metodo .setVisible(false);

L’importo è recuperato tramite Un file che si va a creare al momento del pagamento nella schermata

Precedente , l’intento del file era simulare una webSocket che trasporta informazioni da una pagina all’altra in una web Application, In quanto questo progetto si sarebbe potuto sviluppare anche online.

Se Il pagamento va a Buon fine allora Riceviamo Un messaggio di conferma ricevuto pagamento, viceversa ne riceviamo un altro che ci comunica l’errore.

Una volta effettuato il pagamento le bici sono rese nuovamente disponibili e non è detto che si trovino nello stesso parcheggio.

Il pagamento è effettuato Utilizzando il patter factory method.

### Immagine che contiene testo, schermata, linea, Carattere Descrizione generata automaticamentePattern Factory Method:

### Pattern Factory Method Utilizzato:

Immagine che contiene testo, schermata, diagramma, design

Descrizione generata automaticamente

## DATABASE:

### Immagine che contiene testo, schermata, diagramma, Carattere Descrizione generata automaticamenteRelazionale