



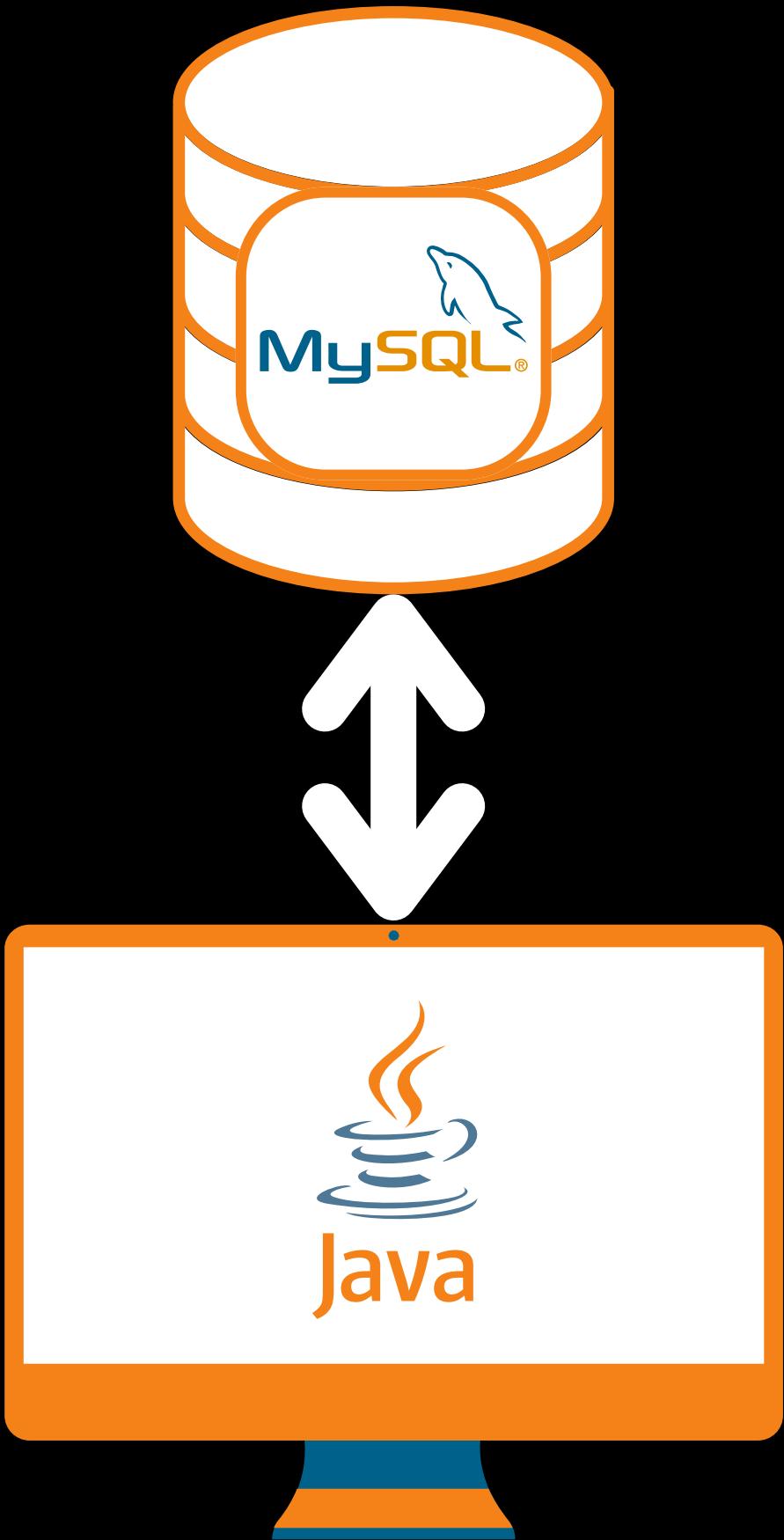
UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARTHENOPE

RestaurAPP

Progetto in Java per l'esame di Programmazione 3/LAB

RUOTOLI - 2322
BIANCAROSA - 2098
MILITERNO - 2454





Le nostre tecnologie

Per lo sviluppo del programma le risorse sono state articolate in questo modo:

- Back-end: sviluppato in Java
- Front-end: sviluppato in Javafx
- Database: Creato con MySql
- Hosting del database: Database SQL

Funzionalità del programma



Il sistema deve prevedere l'accesso sia in modalità amministratore che in modalità utente.

L'amministratore può effettuare le seguenti operazioni:

- Aggiungere o modificare piatti
- Controllare gli utenti registrati
- Visualizzare gli ordini pagati
- Permettere il pagamento degli ordini

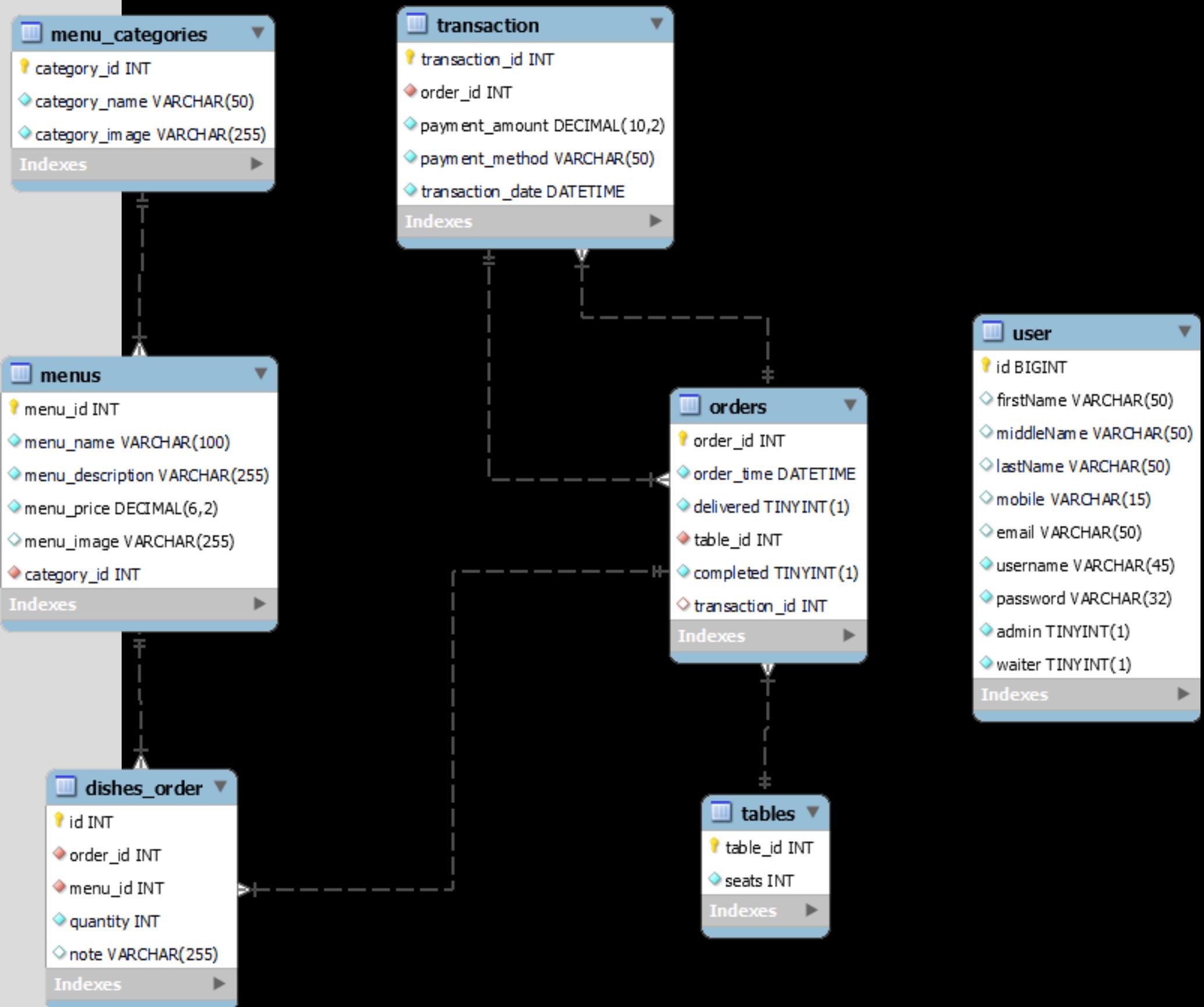
Il Cameriere può effettuare le seguenti operazioni:

- Aggiungere gli ordini abbinati al tavolo

STRUTTURA DEL DATABASE

il database è stato creato seguendo le linee guida del database relazionale MySql ed ha le seguenti tabelle:

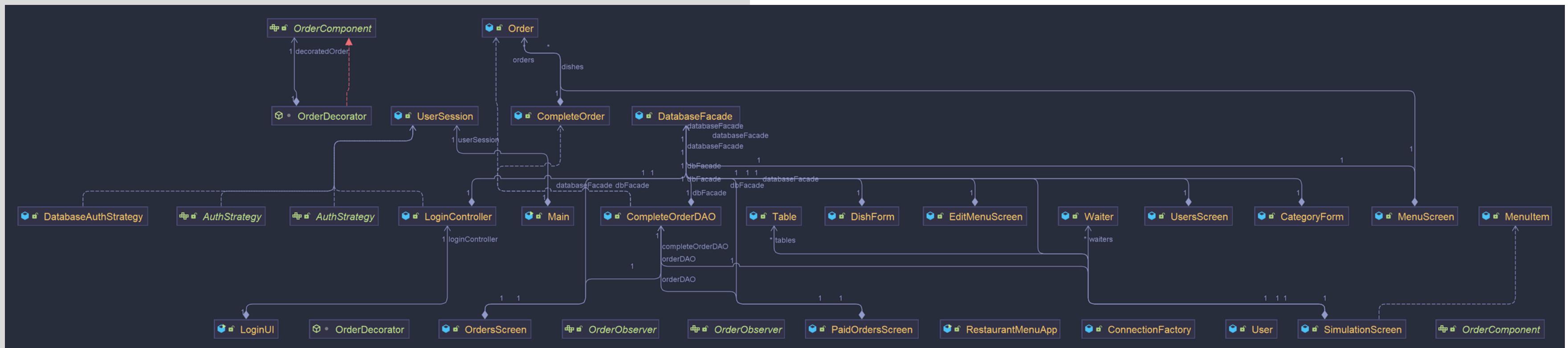
- User mantiene le informazioni riguardo gli utenti
- menu_categories mantiene le informazioni riguardo la categoria
- menus mantiene le informazioni dei piatti;
- dishes_order e orders sono le tabelle degli ordini
- tables mantiene le informazioni sui tavoli, per capire quale tavolo ha effettuato l'ordine
- transaction mantiene le informazioni sul pagamento dell'ordine



STRUTTURA DEL PROGETTO

- AuthStrategy: Interfaccia di scelta Login (Waiter/Admin)
- CompleteOrder: Gestisce completamento ordine (Consegna/ Pagamento)
- MenuItem: Record piatto (nome, id, prezzo)
- Order: Gestisce singolo ordine
- OrderComponent: Interfaccia per note e quantità ordine
- OrderDecorator: Modifica ordine
- OrderObserver: Notifica cambiamenti stato ordine
- UserSession: Gestisce sessione accesso DB
- Waiter: Record del cameriere
- CategoryForm: Form a tendina modificabile dei piatti
- CompleteOrderDAO: Connette ordini DB e App
- ConnectionFactory: Gestisce connessione DB

- Table: Gestisce tavolo
- User: Gestisce utente
- DatabaseAuthStrategy: Autenticazione DB (Waiter/Admin)
- DatabaseFacade: Esegue statement e ottiene risultati DB
- DishForm: Form del singolo piatto
- EditMenuScreen: Modifiche UI del menu
- LoginController: Logica Login
- LoginUI: UI del Login
- Main: Classe principale dell'app
- MenuScreen: UI del menu (piatti, prezzi)
- OrdersScreen: Schermata ordini, notifiche stato ordini
- PaidOrdersScreen: Gestisce ordini pagati
- RestaurantMenuApp: Classe centrale dell'app
- SimulationScreen: UI schermata di simulazione
- UserScreen: UI schermata utente



I nostri ingredienti

Decorator

Aggiungiamo funzionalità ad un oggetto,
senza dover modificare la struttura

Strategy

Famiglia di algoritmi per svolgere un
compito in diversi modi

Factory

Creiamo oggetti nascondendone
l'implementazione all'utente

DAO

Astrazione ed encapsulamento
dell'accesso ai dati

Observer

Notifica automatica dei cambiamenti di
stato dell'oggetto

Facade

Interfaccia semplificata per un gruppo
complesso di classi



DECORATOR

Aggiungi funzionalità senza modificare la ricetta!

Cosa lo
rende
speciale:

Il nostro Decorator permette l'espandibilità del codice dell'oggetto senza alterarne la logica

Qualità
uniche

Implementazione semplice ed efficace

Garanzia
speciale:

Funziona, e lo fa bene

```
abstract class OrderDecorator implements OrderComponent { no usages
    protected final OrderComponent decoratedOrder; 10 usages
    public OrderDecorator(OrderComponent decoratedOrder) { this.decoratedOrder = decoratedOrder; }
    @Override 5 usages
    public int getMenuId() { return decoratedOrder.getMenuId(); }
    @Override 5 usages
    public String getDishName() { return decoratedOrder.getDishName(); }
    @Override 6 usages
    public double getDishPrice() { return decoratedOrder.getDishPrice(); }
    @Override 6 usages
    public int getQuantity() { return decoratedOrder.getQuantity(); }
    @Override 2 usages
    public void incrementQuantity() { decoratedOrder.incrementQuantity(); }
    @Override 4 usages
    public void setQuantity(int quantity) { decoratedOrder.setQuantity(quantity); }
    @Override 2 usages
    public double getTotalPrice() { return decoratedOrder.getTotalPrice(); }
    @Override 3 usages
    public String getNotes() { return decoratedOrder.getNotes(); }
    @Override 4 usages
    public void setNotes(String notes) { decoratedOrder.setNotes(notes); }
}
```

STRATEGY

Scegli il tuo piatto nel modo migliore

Speciale perché:

Questa miscela può essere utilizzata per insaporire proteine e verdure.

Qualità uniche:

Questo mix di spezie comprende nove ingredienti la cui combinazione è perfetta.

Garanzia speciale:

Le dosi sono state provate e testate per trovare la miscela più saporita.

```
public class DatabaseAuthStrategy implements AuthStrategy {
    @Override 1 usage
    public UserSession authenticate(String userId, String password) {
        UserSession userSession = null;
        DatabaseFacade dbFacade = new DatabaseFacade();
        try {
            dbFacade.openConnection();
            Connection conn = dbFacade.getConnection();
            String query = "SELECT admin FROM user WHERE username = ? AND password = ?";
            try (PreparedStatement preparedStatement = conn.prepareStatement(query)) {
                preparedStatement.setString(parameterIndex: 1, userId);
                preparedStatement.setString(parameterIndex: 2, password);

                ResultSet resultSet = preparedStatement.executeQuery();
                if (resultSet.next()) {
                    boolean isAdmin = resultSet.getBoolean(columnLabel: "admin");
                    userSession = UserSession.getInstance(userId, isAdmin);
                }
            } catch (SQLException e) {
                e.printStackTrace();
            } finally {
                dbFacade.closeConnection();
            }
        }
        return userSession;
    }
}
```

```
public interface AuthStrategy { 1 implementation
    UserSession authenticate(String userId, String password); 1 usage 1 implementation
}
```

FACTORY

Quando ordini un piatto viene istanziato in base a cosa ordini

```
public class ConnectionFactory {  
    private static final String DB_URL = "jdbc:mysql://localhost:3306/restaurant"; 1 usage  
    private static final String DB_USER = "root"; 1 usage  
    private static final String DB_PASSWORD = "562656";| 1 usage  
    public static Connection createConnection() throws SQLException { 1 usage  
        return DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);  
    }  
}
```

Speciale perché:

La ricetta del piatto cambia istanza in base alla necessità

Qualità uniche:

Ricetta nascosta al cliente, vedrà solo l'istanza implementata

Garanzia speciale:

L'implementazione è unica per ogni piatto

DAO

Il piatto con ingredienti più accessibili

```
public class CompleteOrderDAO {  
    private DatabaseFacade dbFacade; 21 usages  
    public CompleteOrderDAO() { this.dbFacade = new DatabaseFacade(); }  
    private List<OrderObserver> observers = new ArrayList<>(); 1 usage  
    public void addObserver(OrderObserver observer) { observers.add(observer); }  
    public List<CompleteOrder> getAllNonCompletedOrders() {...}  
    public List<CompleteOrder> getAllCompletedOrders() {...}  
    public void updateOrderStatus(int orderId, boolean delivered) {...}  
    public void deleteOrder(int orderId) {...}  
    public void deleteDish(int orderId, int menuId) {...}  
    public int processPaymentTransaction(int orderId, String paymentMethod) {...}  
    private List<Order> getDishesForOrder(int orderId, Connection conn) throws SQLException
```

Speciale perché:

Accesso semplificato agli ingredienti singoli che lo compongono

Qualità uniche:

Gli ingredienti del piatto sono accessibili unicamente dal piatto stesso

Garanzia speciale:

Ingredienti selezionati a mano ed inseriti con cura nel Pattern

OBSERVER

Il nostro mix di osservatori, in una pratica confezione

```
public interface OrderObserver { 1 implementation
    void onOrderStatusChanged(int orderId, boolean delivered); 1 usage 1 implementation
    void onOrderDeleted(int orderId); 1 usage 1 implementation
    void onDishDeleted(int orderId, int menuId); 1 usage 1 implementation
    void onPaymentProcessed(int orderId, int transactionId, String paymentMethod, double amountReceived);
}
```

Speciale perché:

Il piatto permette la notifica dei cambiamenti

Qualità uniche:

Avvisa se viene modificata la sua ricetta

Garanzia speciale:

La notifica è segnalata al cameriere immediatamente

FACADE

Un piatto semplice che nasconde una ricetta complessa

Speciale perché:

Questo delizioso pattern semplifica tutti gli ingredienti migliori della ricetta.

Qualità uniche:

L'unione dell'implementazione delle classi fornisce un gusto delicato tramite l'uso di una patina al gusto di interfaccia

Garanzia speciale:

Implementato senza ChatGPT

```
public class DatabaseFacade {  
    private Connection connection; 8 usages  
    public Connection openConnection() throws SQLException {...}  
    public void closeConnection() {...}  
    public Connection getConnection() { return connection; }  
    public ResultSet executeQuery(String query) throws SQLException {...}  
    public ResultSet executeQuery(String query, int param) throws SQLException {...}  
    public void executeUpdate(String query, Object... params) throws SQLException {...}  
}
```



Come effettuare un ordine

MANGIARE NEI NOSTRI
LOCALI È FACILE E
VELOCE

Punto 1

Effettua un ordine al tavolo
attendendo i nostri camerieri
esplcitando le aggiunte
necessarie.

Punto 2

La cucina preparerà il tuo ordine
ed il cameriere lo consegnerà al
tuo tavolo!

Punto 3

Il cameriere confermerà alla
cucina la corretta consegna del
tuo ordine, e tu potrai gustarlo!

Punto 4

Recati in cassa a pagare, e
conserva lo scontrino!

SISTEMA DI AUTOMAZIONE

Simulazione del software

Grazie a questa simulazione possiamo testare il corretto funzionamento del software

I tavoli rossi sono quelli che non hanno ancora effettuato un ordine

I tavoli gialli sono quelli che aspettano che l'ordine sia pronto

I tavoli verdi sono quelli che stanno consumando l'ordine per poi tornare rossi

