

# Ingegneria del Software

Presentazione dell'elaborato

Paolo Formentelli  
Paolo Giacomo Vicardi



UNIVERSITÀ  
DEGLI STUDI  
DI BRESCIA

# Indice:

• Principio di separazione modello-vista	<u>2-8</u>
• GRASP Information Expert	<u>9</u>
• GRASP Creator	<u>10</u>
• SOLID Open-closed	<u>11</u>
• SOLID Interface Segregation	<u>12-14</u>
• GoF Simple Factory	<u>15-16</u>
• Testing Black-Box	<u>17</u>
• Refactoring Move Method	<u>18-20</u>

# MVC: SCELTA PROGETTUALE

- **OPZIONE 1:** view unica per tutte le classi
- **OPZIONE 2:** separazione delle view con compiti specifici

# MVC: SEPARAZIONE DELLE VIEW

## Vantaggi:

- Alta coesione
- Manutenzione facilitata
- Facilità di test
- Scalabilità

```
public interface View {  
  
    default void showMenu() {}  
  
    default void showCurrentDate() {  
        System.out.println("\n" + LocalDate.now().format(DateTimeFormatter.ofLocalizedDate(FormatStyle.FULL)).toUpperCase());  
    }  
  
    default void showMessage(String message) { System.out.println(message); }  
}
```

```
public class StorageView implements View {  
  
    @Override  
    public void showMenu() {  
        showMessage("1 - Visualizza registro");  
        showMessage("2 - Aggiungi merci al registro");  
        showMessage("3 - Crea lista della spesa");  
        showMessage("0 - Menu principale");  
    }  
  
    public void showStorageSize(int size) {  
        newLine();  
        showMessage("Ci sono " + size + " ingredienti nel magazzino");  
    }  
  
    public void showDrinksSize(int size) {  
        newLine();  
        showMessage("Ci sono " + size + " bevande nel magazzino");  
    }  
}
```

```
public class ReservationsView implements View {  
  
    @Override  
    public void showMenu() {  
        showMessage("1 - Visualizza prenotazioni");  
        showMessage("2 - Aggiungi prenotazione");  
        showMessage("3 - Cancella prenotazione");  
        showMessage("0 - Menu principale");  
    }  
  
    public void showSize(int size) {  
        newLine();  
        showMessage("Ci sono " + size + " prenotazioni");  
    }  
  
    public void showNoParametersMessage() {  
        showMessage("I parametri del ristorante non sono ancora stati inizializzati dal gestore");  
    }  
}
```

# MVC: SEPARAZIONE MODELLO-VISTA

```
public class Manager {  
    public static void initialize() throws Exception {  
        if (n_client == 0) {  
            System.out.println("\nBenvenuto! Prima di iniziare, inserisci i seguenti parametri del ristorante");  
            n_client = InputDati.leggiInteroPositivo( messaggio: "Numero di posti a sedere: ");  
            workload_for_client = InputDati.leggiInteroPositivo( messaggio: "Carico di lavoro per persona (un valore di circa 40 indica un carico di lavoro normale): ");  
            drinks = InputDati.leggiInteroNonNegativo( messaggio: "Consumo pro capite di bevande: ");  
            extras = InputDati.leggiInteroNonNegativo( messaggio: "Consumo pro capite di generi alimentari extra: ");  
            workload_for_restaurant = workload_for_client * (n_client + ((n_client * 20)/100));  
  
            parametersWriter(n_client, workload_for_client, drinks, extras);  
        }  
  
        int option;  
        do {  
            List<Recipe> recipes = recipeReader();  
            List<Menu> thematic_menu_list = menuReader();  
            List<Ingredient> drinks = xmlReader( drink: true);  
            List<Ingredient> extras = xmlReader( drink: false);  
  
            System.out.println("\n" + LocalDate.now().format(DateTimeFormatter.ofLocalizedDate(FormatStyle.FULL)).toUpperCase());  
            System.out.println("1 - Gestisci Ricette");  
            System.out.println("2 - Gestisci Bevande");  
            System.out.println("3 - Gestisci Extra");  
            System.out.println("4 - Gestisci Menù Tematici");  
            System.out.println("5 - Visualizza proprietà Ristorante");  
            System.out.println("0 - Menu principale");  
  
            option = InputDati.leggiIntero( messaggio: "Scegli: ", minimo: 0, massimo: 5);  
            System.out.print("\n");  
  
            switch (option) {  
                //Gestisci ricette  
                case 1 -> {  
  
                    System.out.println("1 - Visualizza ricetta");  
                    System.out.println("2 - Aggiungi ricetta");  
                    System.out.println("3 - Rimuovi ricetta");  
                    int option_1 = InputDati.leggiIntero( messaggio: "Scegli: ", minimo: 1, massimo: 3);  
                    System.out.print("\n");  
  
                    switch (option_1) {  
                        //Visualizza ricette
```

Utilizzo sbagliato del modello MVC nella classe Manager della prima parte dell'elaborato

```
public class Manager { 10 usages  Paolo Vicardi +1
```

```
    public static final Date[] RESTAURANT_OPENING_PERIOD = {new Date()};
    private final ManagerReader reader; 5 usages
    private final ManagerUpdater updater; 7 usages
    private int[] parameters; 5 usages

    public Manager(ManagerReader reader, ManagerUpdater updater) {
        this.reader = reader;
        this.updater = updater;
        this.parameters = reader.parametersReader();
    }
```

```
public class ManagerController { 2 usages  Paolo Vicardi
```

```
    private final Manager manager; 50 usages
    private final ManagerView managerView; 87 usages
```

```
    public ManagerController(Manager manager, ManagerView managerView) {
        this.manager = manager;
        this.managerView = managerView;
    }
```

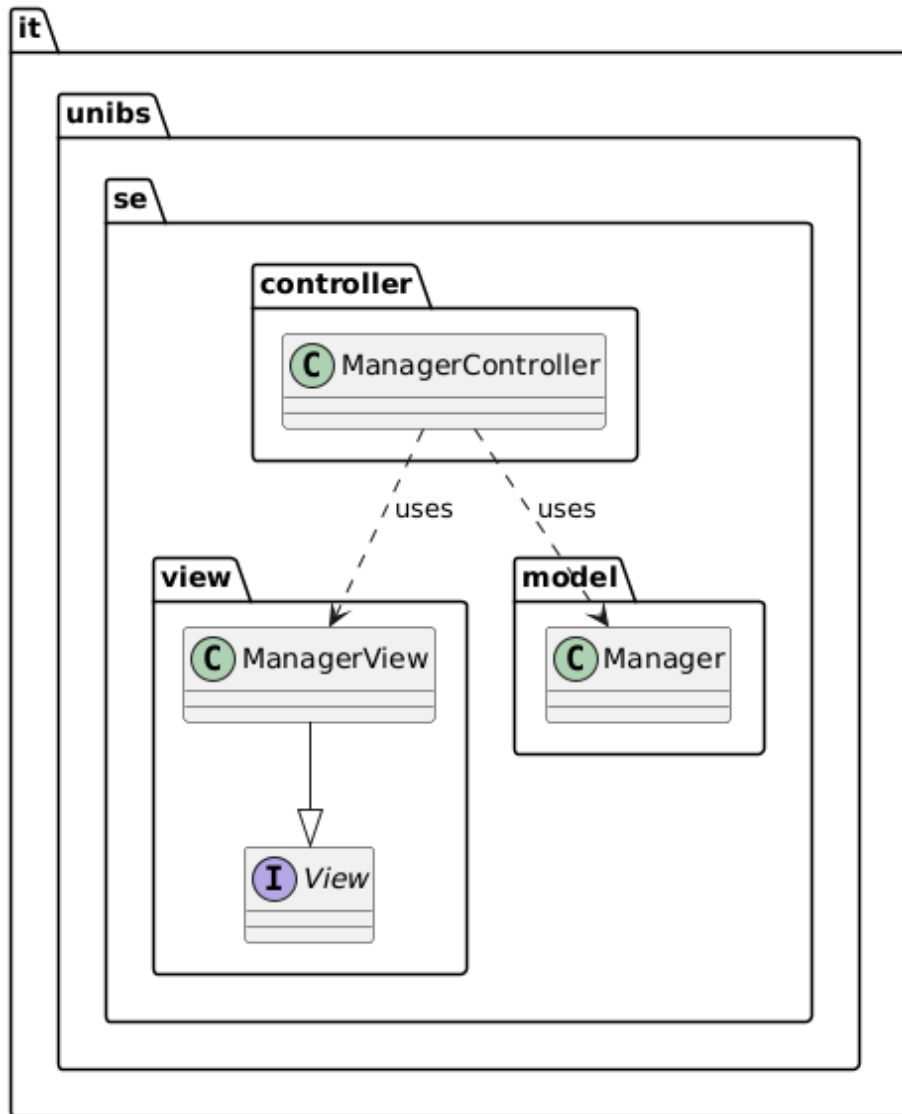
```
public class ManagerView implements View { 5 usages  Paolo Vicardi
```

```
    @Override 4 usages  Paolo Vicardi
```

```
    public void showMenu() {
        showMessage("1 - Gestisci Ricette");
        showMessage("2 - Gestisci Bevande");
        showMessage("3 - Gestisci Extra");
        showMessage("4 - Gestisci Menù Tematici");
        showMessage("5 - Visualizza proprietà Ristorante");
        showMessage("0 - Menu principale");
    }
```

Separazione MVC della classe Manager

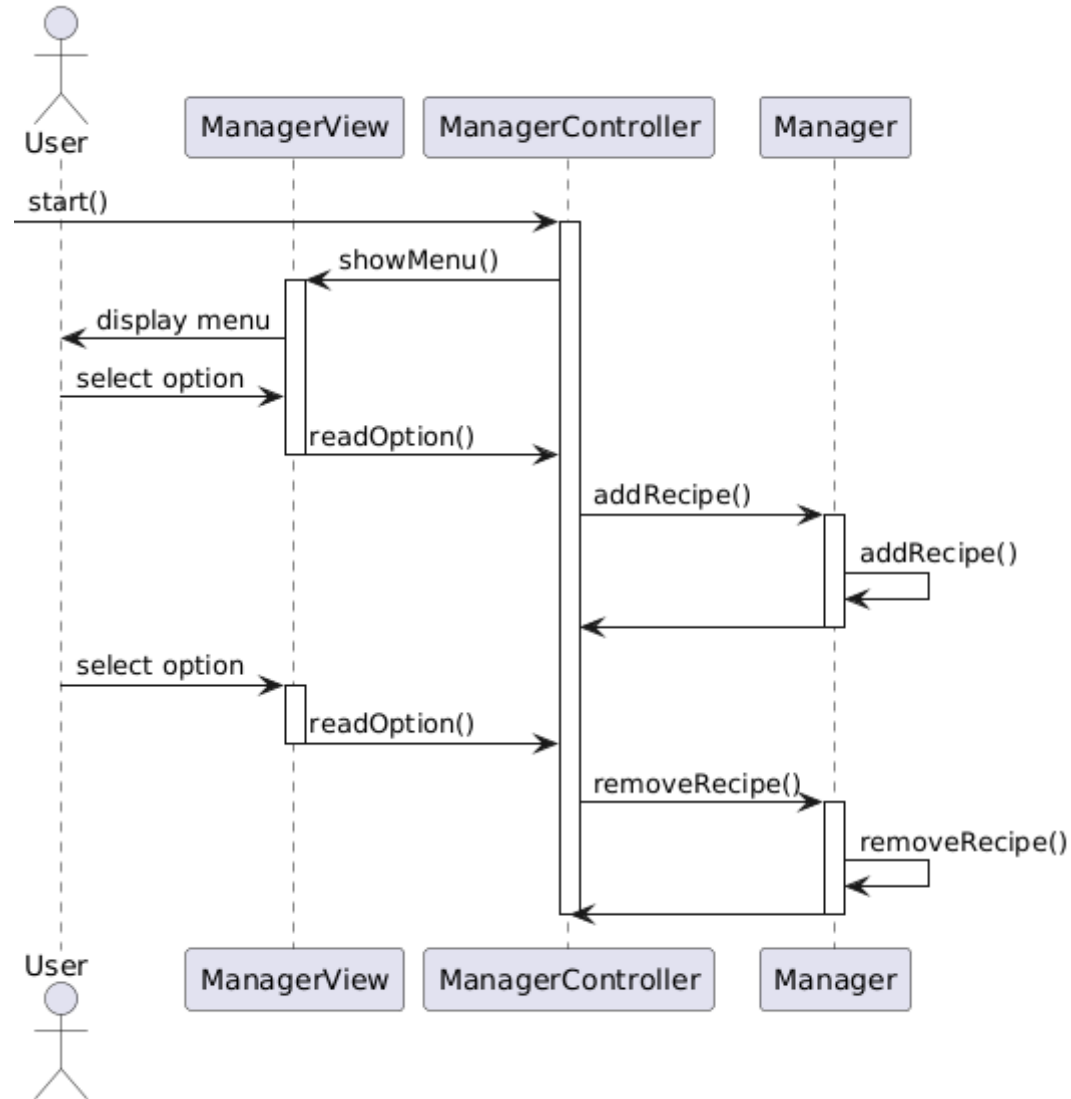
# MVC: DIPENDENZE



Le dipendenze vengono iniettate direttamente dal costruttore evitando che la view acceda direttamente alla logica di business:

```
Manager manager = new Manager(reader, updater);  
ManagerView managerView = new ManagerView();  
ManagerController managerController = new ManagerController(manager, managerView);
```

# MVC: SSD MANAGER





# GRASP: PATTERN VALUTATIVI

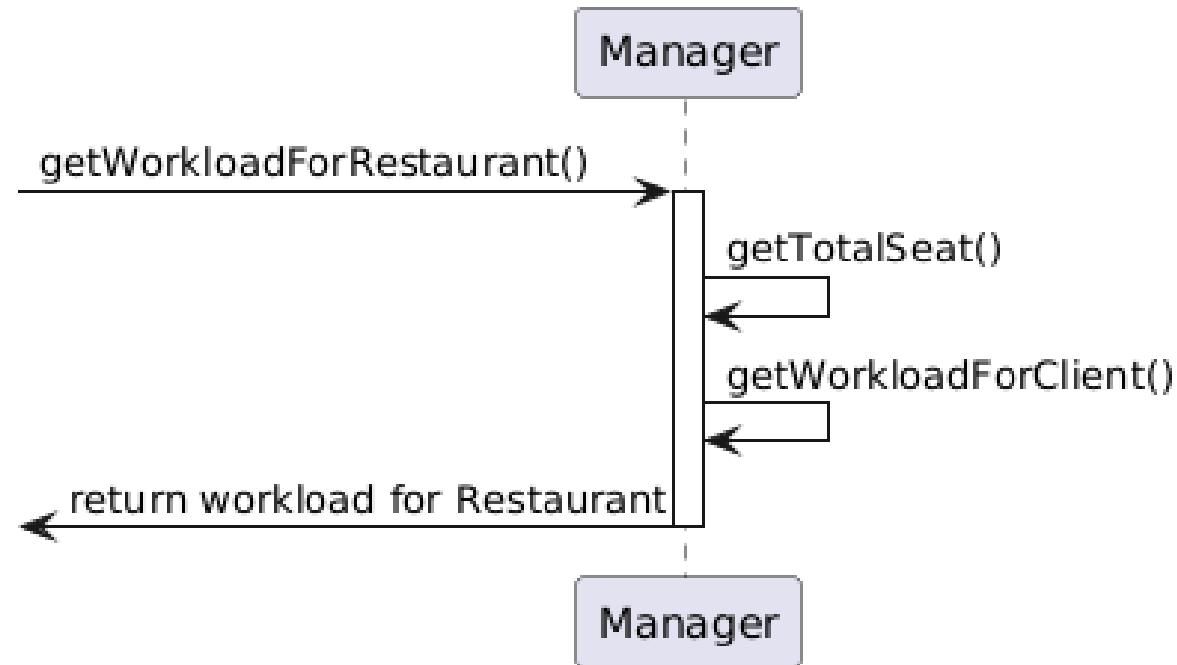
```
public class ManagerController { 2 usages  👤 Paolo Vicardi
    private final Manager manager; 50 usages
    private final ManagerView managerView; 87 usages

    public ManagerController(Manager manager, ManagerView
        this.manager = manager;
        this.managerView = managerView;
    }
```

Low Coupling + High Cohesion

# PATTERN GRASP: INFORMATION EXPERT

```
public int getWorkloadForRestaurant() { return getWorkloadForClient() * (getTotalSeats() * 12 / 10); }
```



# PATTERN GRASP: CREATOR

```
private Recipe handleNewRecipeRequest() { 1 usage  👤 Paolo Vicardi
    managerView.newLine();
    String recipe_name = managerView.readNotEmptyRecipeName();
    int id = manager.calculateNewId(manager.getRecipes());
    List<GenericIngredient> ingredients = handleRecipeIngredientsRequest();
    int workload = ingredients.size() * 2;
    Date[] period = handleRecipePeriodRequest();
    return new Recipe(recipe_name, id, workload, ingredients, period[0], period[1]);
}
```

Questo metodo della classe ManagerController ottiene tutti i dati per la creazione di una nuova ricetta

# PATTERN SOLID: OPEN CLOSED

```
public interface Identifiable {  
    int getId(); 4 implementations  
    void setId(int id); 4 implementer  
}
```

```
public <T extends Identifiable> int calculateNewId(List<T> items) {  
    int id = 1;  
    if (!items.isEmpty()) {  
        id = items.get(items.size() - 1).getId() + 1;  
    }  
    return id;  
}
```

Il metodo CalculateNewId() può essere esteso a nuovi tipi di dati che implementino Identifiable senza richiedere modifiche al codice esistente

# PATTERN SOLID: INTERFACE SEGREGATION

```
30 public class Xml { 4 usages ± Paolo Vicardi +1 *
31
32     private static final String RESERVATIONS_PATH = "restaurant/src/it/unibs/xml/Reservations.xml"; 3 usages
33     private static final String RECIPES_PATH = "restaurant/src/it/unibs/xml/Ricettario.xml"; 5 usages
34     private static final String DRINKS_PATH = "restaurant/src/it/unibs/xml/Drinks.xml"; 3 usages
35     private static final String EXTRAS_PATH = "restaurant/src/it/unibs/xml/Extras.xml"; 3 usages
36     private static final String HOLIDAYS_PATH = "restaurant/src/it/unibs/xml/Holidays.xml"; 1 usage
37     private static final String TEMATIC_MENU_PATH = "restaurant/src/it/unibs/xml/TematicMenu.xml"; 3 usages
38     private static final String STORAGE_PATH = "restaurant/src/it/unibs/xml/Storage.xml"; 3 usages
39
40 @ > public static List<Recipe> recipeReader() {...}
77
78 @ > public static List<Ingredient> xmlReader(boolean drink) {...}
104
105 @ > public static List<Date> holidaysReader() {...}
127
128 @ > public static List<Menu> menuReader() {...}
170
171 @ > public static List<Ingredient> storageReader() {...}
196
197 > public static void recipeWriter() throws Exception {...}
262
263 > public static void xmlWriter(boolean drinks) throws Exception {...}
306
307 @ > public static List<Reservation> reservationReader() {...}
382
383 @ > public static void reservationWriter(List<Reservation> reservations, List<Date> holidays) throws Exception {...}
590
591 > public static void elementDeleter(int option, int idValue) {...}
645
646 > public static void menuWriter() throws Exception {...}
715
716 > public static void attributeUpdater(int id) throws Exception {...}
749
750 @ > public static void storageUpdater(int new_id, List<Ingredient> storage) throws Exception {...}
789
790 @ > public static int[] parametersReader() {...}
821
822 > public static void parametersWriter(int seats_value, int workload_value, int drinks_value, int extras_value) throws Exception {...}
845
846 }
847
```

Prima del refactoring

# PATTERN SOLID: INTERFACE SEGREGATION

```
25 public class DatabaseUpdater implements ManagerUpdater, ReservationUpdater, StorageUpdater { 4 usages ± Paolo Vicardi *
26
27     @Override 1 usage ± Paolo Vicardi
28     public void parametersWriter(int seats_value, int workload_value, int drinks_value, int extras_value) throws Exception {...}
51
52     @Override 1 usage ± Paolo Vicardi
53     public void recipeWriter(Recipe recipe) throws Exception {...}
83
84     @Override 1 usage ± Paolo Vicardi
85     public void drinkWriter(IdentifiableIngredient drink) throws Exception {...}
94
95     @Override 1 usage ± Paolo Vicardi
96     public void extraWriter(IdentifiableIngredient extra) throws Exception {...}
104
105     @Override 1 usage ± Paolo Vicardi
106     public void menuWriter(Menu new_menu) throws Exception {...}
134
135     @Override 1 usage ± Paolo Vicardi
136     public void reservationWriter(Reservation reservation) throws Exception {...}
175
176     @Override 1 usage ± Paolo Vicardi
177     public void storageWriter(GenericIngredient new_ingredient, int new_id) throws Exception {...}
194
195     @Override 1 usage ± Paolo Vicardi
196     public void updateIngredientQuantity(String type, int id, float increment) throws Exception {...}
224
225     @Override 3 usages ± Paolo Vicardi
226     public void deleteElement(String element_name, int idValue) {...}
267
268     @ private static void appendIngredient(File xmlFile, Document doc, Element new_ingredient, String name, int id, float quantity) thr
278
279     @ private static void appendALaCarteOrder(Document doc, Element new_reservation, List<Integer> ordered_dishes_ids, int number_of_pe
296
297     @ private static void appendThemedMenuOrder(Document doc, Element new_reservation, int ordered_menu_id, int number_of_people) {...}
312
313     private static void updateDatabase(File xmlFile, Document doc) throws IOException, TransformerException {...}
320
321 }
```

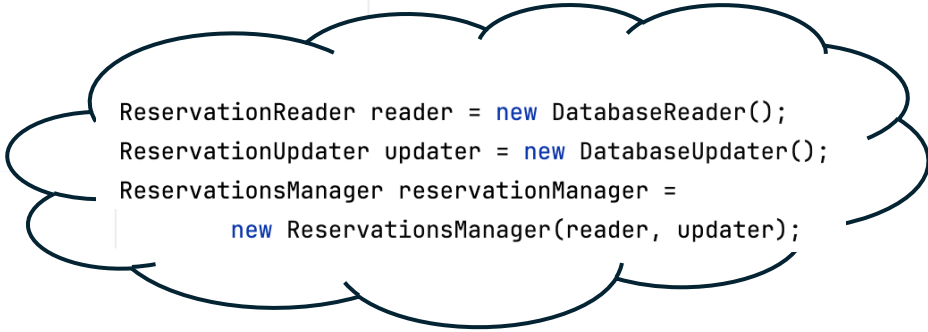
```
19 public class DatabaseReader implements ManagerReader, ReservationReader, StorageReader { 4 usages ± Paolo Vicardi
20
21     private static final String RESERVATIONS_PATH = "restaurant/src/it/unibs/xml/Reservations.xml"; 1 usage
22     private static final String RECIPES_PATH = "restaurant/src/it/unibs/xml/Ricettario.xml"; 1 usage
23     private static final String DRINKS_PATH = "restaurant/src/it/unibs/xml/Drinks.xml"; 1 usage
24     private static final String EXTRAS_PATH = "restaurant/src/it/unibs/xml/Extras.xml"; 1 usage
25     private static final String HOLIDAYS_PATH = "restaurant/src/it/unibs/xml/Holidays.xml"; 1 usage
26     private static final String THEMED_MENU_PATH = "restaurant/src/it/unibs/xml/ThemedMenu.xml"; 1 usage
27     private static final String STORAGE_PATH = "restaurant/src/it/unibs/xml/Storage.xml"; 1 usage
28     private static final String PARAMETERS_PATH = "restaurant/src/it/unibs/xml/Parameters.xml"; 1 usage
29
30     @Override 5 usages ± Paolo Vicardi
31     public int[] parametersReader() {...}
62
63     @Override 6 usages ± Paolo Vicardi
64     public List<Recipe> recipesReader() {...}
95
96     @Override 3 usages ± Paolo Vicardi
97     public List<IdentifiableIngredient> drinksReader() {...}
121
122     @Override 3 usages ± Paolo Vicardi
123     public List<IdentifiableIngredient> extrasReader() {...}
147
148     @Override 3 usages ± Paolo Vicardi
149     public List<Menu> themedMenusReader() {...}
188
189     @Override 1 usage ± Paolo Vicardi
190     public List<Date> holidaysReader() {...}
212
213     @Override 2 usages ± Paolo Vicardi
214     public List<Reservation> reservationsReader() {...}
285
286     @Override 2 usages ± Paolo Vicardi
287     public List<GenericIngredient> storageReader() {...}
305
306     private void ingredientsReader(List<GenericIngredient> storage, NodeList ingredientList) {...}
315
316 }
```

# PATTERN SOLID: INTERFACE SEGREGATION

```
public interface ReservationUpdater {  
    void reservationWriter(Reservation reservation) throws Exception;  
    void deleteElement(String type, int id) throws Exception;  
}
```

```
public interface ManagerUpdater {  
    void parametersWriter(int totalSeats, int workloadForClient, int drinksPerPerson, int extrasPerPerson) throws Exception;  
    void recipeWriter(Recipe recipe) throws Exception;  
    void drinkWriter(IdentifiableIngredient drink) throws Exception;  
    void extraWriter(IdentifiableIngredient extra) throws Exception;  
    void menuWriter(Menu menu) throws Exception;  
    void deleteElement(String type, int id) throws Exception;  
}
```

```
public interface StorageUpdater {  
    void storageWriter(GenericIngredient ingredient, int id) throws Exception;  
    void updateIngredientQuantity(String type, int id, float increment) throws Exception;  
}
```



```
ReservationReader reader = new DatabaseReader();  
ReservationUpdater updater = new DatabaseUpdater();  
ReservationsManager reservationManager =  
    new ReservationsManager(reader, updater);
```

Non costringe nessuno a dipendere da metodi che non usa

# PATTERN GoF: (SIMPLE) FACTORY

```
public class IngredientFactory { 4 usages

    public static IdentifiableIngredient createDrink(String name, float quantity_in_stock, int id) { 2 usages  Paolo Vicardi
        return new IdentifiableIngredient(new GenericIngredient(name, quantity_in_stock, unit_of_measure: "bottiglie"), id);
    }

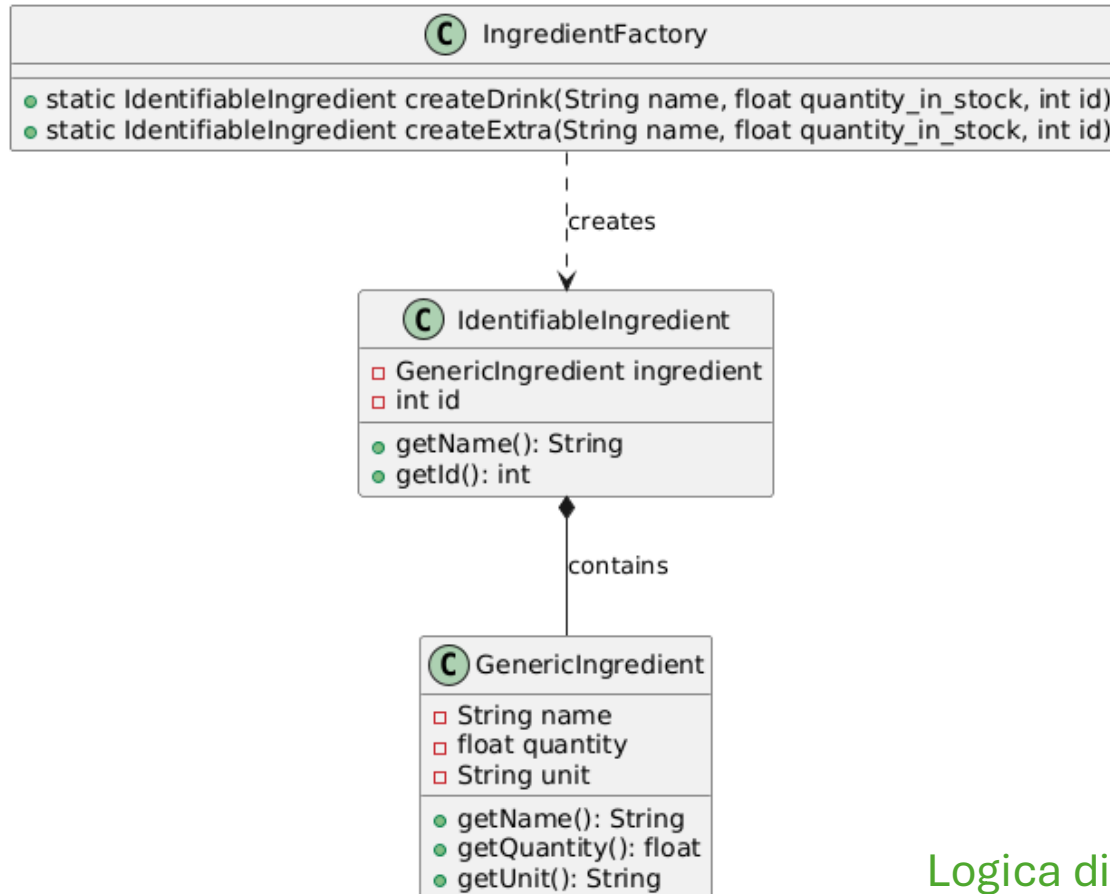
    public static IdentifiableIngredient createExtra(String name, float quantity_in_stock, int id) { 2 usages  Paolo Vicardi
        return new IdentifiableIngredient(new GenericIngredient(name, quantity_in_stock, unit_of_measure: "unità"), id);
    }
}
```

Metodo centralizzato per creazione di oggetti `IdentifiableIngredient` senza esporre logica di costruzione. Se la logica cambia modifichiamo solo `IngredientFactory`.



# PATTERN GoF: (SIMPLE) FACTORY

Simple Factory: IngredientFactory



Logica di creazione separata da logica di business.

# TESTING: BLACK-BOX

DATA	ENTER DAY:	*accepts value 1 to 31		
BOUNDARY VALUE ANALYSIS			WHITTAKER TEST:	
Invalid (min-1)	Valid (min, +min, -max, +max)	Invalid (max+1)	30/02	31/02
			31/04	31/06
0	1,2,30,31	32	31/09	31/11
			29/02 per anni non bisestili	
DATA	ENTER MONTH:	*accepts value 1 to 12		
BOUNDARY VALUE ANALYSIS				
Invalid (min-1)	Valid (min, +min, -max, +max)	Invalid (max+1)		
0	1,2,11,12	13		

# REFACTORING: MOVE METHOD

```
13 public class Reservation { 14 usages ± paoloformentelli +1
72
73 > public void printReservation() {...}
106
107 public static void deleteReservation(List<Reservation> reservations) { 2 usages ± paoloformentelli +1
108     reservations = reservationReader();
109     for (Reservation reservation : reservations)
110         reservation.printReservation();
111     int delete_id;
112     delete_id = InputDati leggiInteroConMinimo( messaggio: "\nInserisci l'id della prenotazione da eliminare: ", minimo: 0);
113     for (Reservation reservation : reservations) {
114         if (reservation.getReservation_id() == delete_id) {
115             reservations.remove(reservation);
116             elementDeleter( option: 4, delete_id);
117             System.out.println("Prenotazione eliminata con successo!");
118             break;
119         }
120     }
121 }
```

Metodo deleteReservation() nella classe Reservation

# REFACTORING: MOVE METHOD

```
public class ReservationsManager { 4 usages ± Paolo Vicardi +2

    private final ReservationReader reader; 7 usages
    private final ReservationUpdater updater; 3 usages

    public ReservationsManager(ReservationReader reader, ReservationUpdater updater) { 1 usage ± Paolo Vicardi
        this.reader = reader;
        this.updater = updater;
    }

    public List<Reservation> getReservations() { return reader.reservationsReader(); } 9 usages ± Paolo Vicardi
    public List<Recipe> getRecipes() { return reader.recipesReader(); } 4 usages ± Paolo Vicardi
    public List<Menu> getMenus() { return reader.themedMenusReader(); } 3 usages ± Paolo Vicardi
    public List<Date> getHolidays() { return reader.holidaysReader(); } 1 usage ± Paolo Vicardi
    public int getTotalSeats() { return reader.parametersReader()[0]; } 3 usages ± Paolo Vicardi
    public int getWorkloadForClient() { return reader.parametersReader()[1]; } 3 usages ± Paolo Vicardi

    public void addReservation(Reservation reservation) { 1 usage ± Paolo Vicardi
        try {
            updater.reservationWriter(reservation);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    public boolean deleteReservation(int delete_id) { 1 usage ± Paolo Vicardi +1
        List<Reservation> reservations = getReservations();
        for (Reservation reservation : reservations) {
            if (reservation.getId() == delete_id) {
                reservations.remove(reservation);
                try {
                    updater.deleteElement( type: "reservation", delete_id);
                } catch (Exception e) {
                    throw new ConvertException("Errore nella rimozione della prenotazione", e);
                }
                return true;
            }
        }
        return false;
    }
}
```

Metodo deleteReservation() nella classe ReservationManager

# REFACTORING: ANALISI MOVE METHOD

1. Attributi e metodi utilizzati
2. Utilizzi del metodo:
3. Dichiarazione del metodo in ReservationManager
4. Sistemazione metodi utilizzati in deleteReservation()
5. Compilazione e test
6. Rimozione del vecchio metodo

