

Intervista

Struttura Intervista

Brainstorming
+
Analisi di
Mercato

Sviluppo Idee
Innovative

Strutturazione
Intervista

- **Chi:** per inquadrare meglio la figura del committente.
- **Che cosa:** per comprendere nel dettaglio le esigenze del committente e la sua idea di applicazione finale.
- **Quando:** per definire con precisione le tempistiche relative alla finalizzazione del progetto. Il tempo a disposizione determina una selezione dei moduli innovativi da sviluppare.

Risultati

Moduli Core

- Pop-up stores
- Magazzino snello
- Vestirsi con la realtà aumentata
- Go circular!

Moduli Pilota

- Algoritmo di consigli all'acquisto human-driven
- Sartoria dedicata e confezionamento su misura
- Social marketing

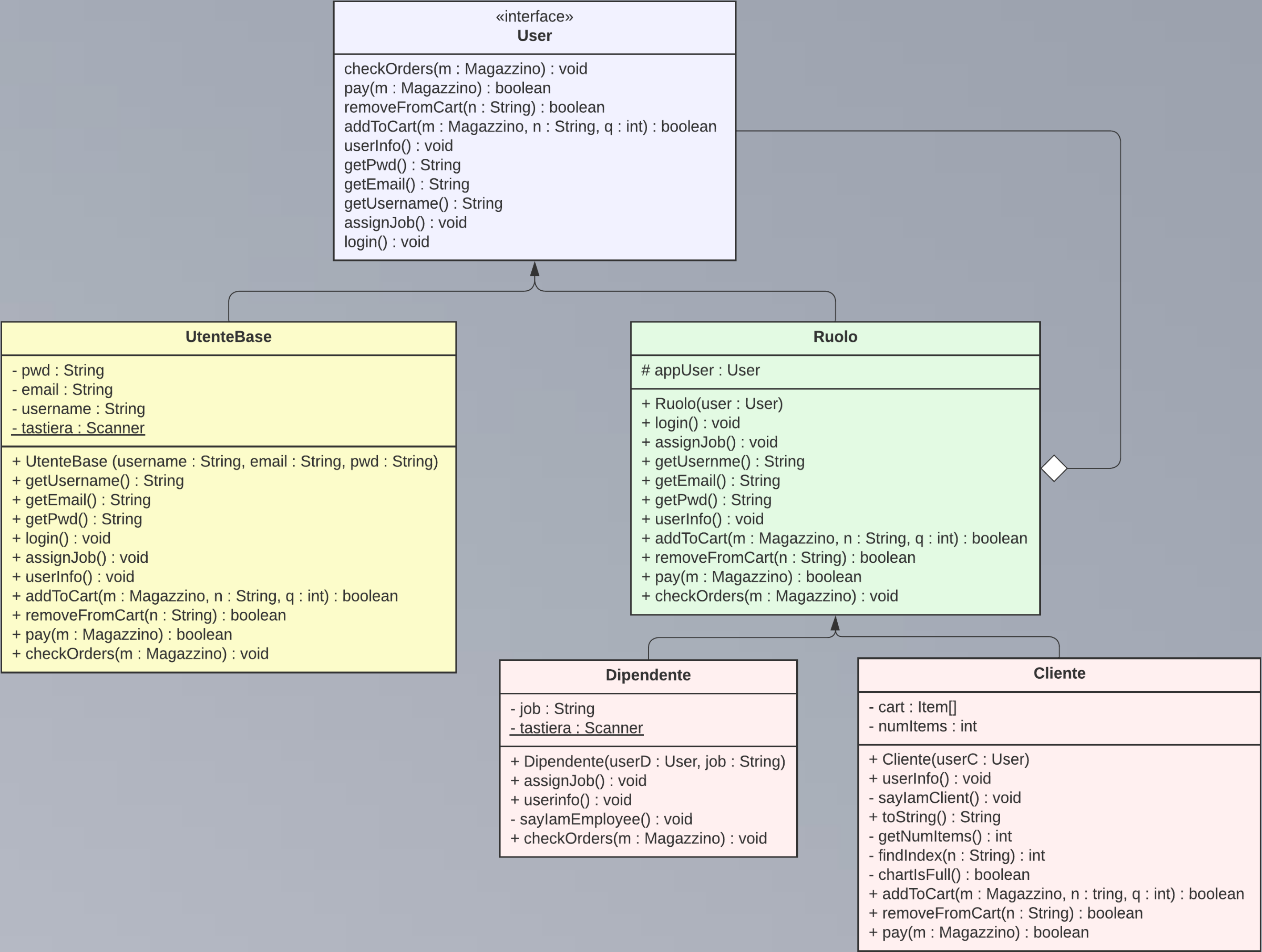
Il Pattern Decorator

Vantaggi:

- È più flessibile della ereditarietà statica (multipla)
- Approccio “pay as you go” aggiungendo responsabilità

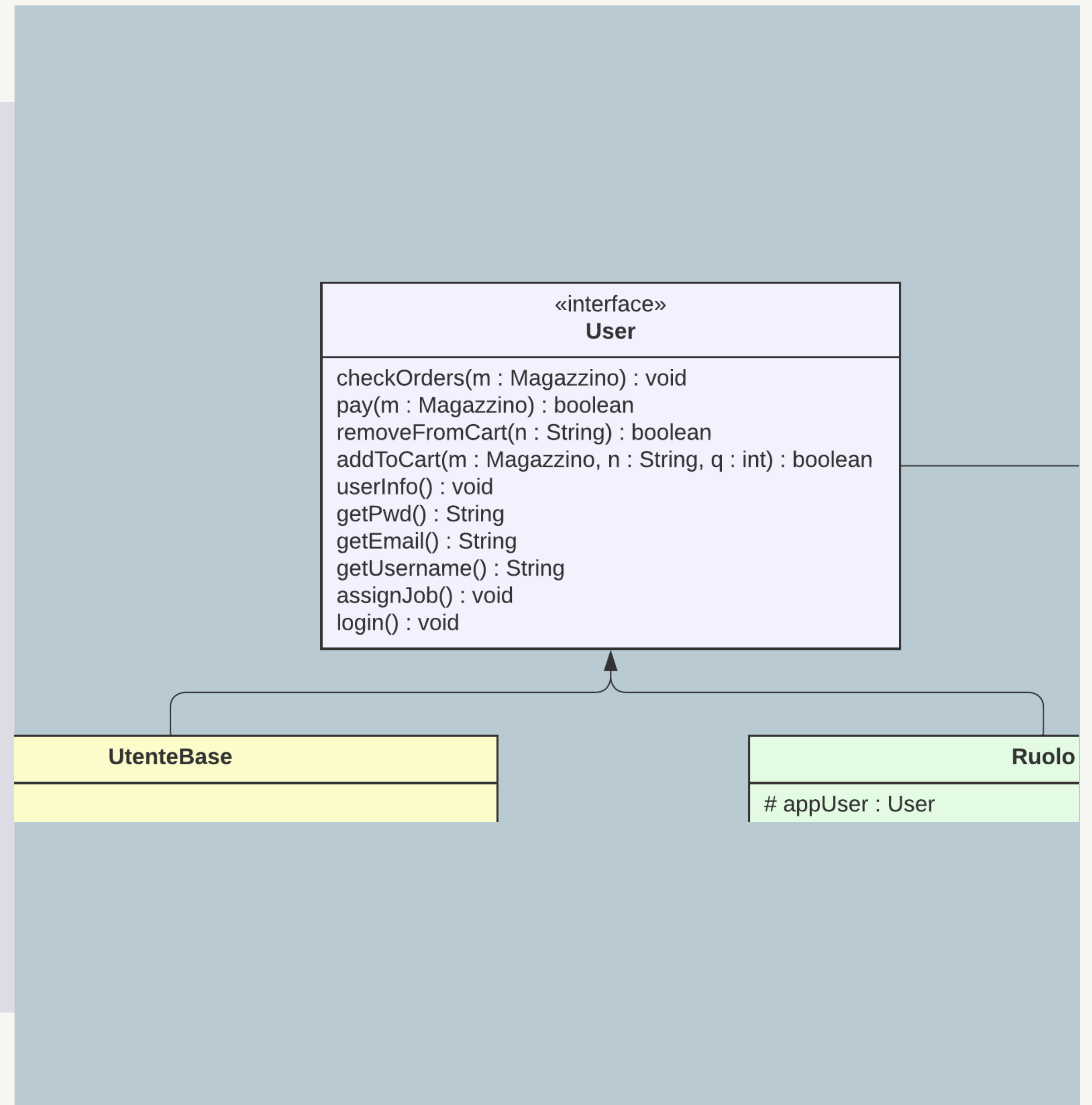
Svantaggi:

- Il decorator e i suoi component non sono identici (==)
- Genera tanti “piccoli” oggetti simili tra di loro



Component

Definisce l'interfaccia per gli oggetti a cui verranno aggiunte responsabilità in modo dinamico



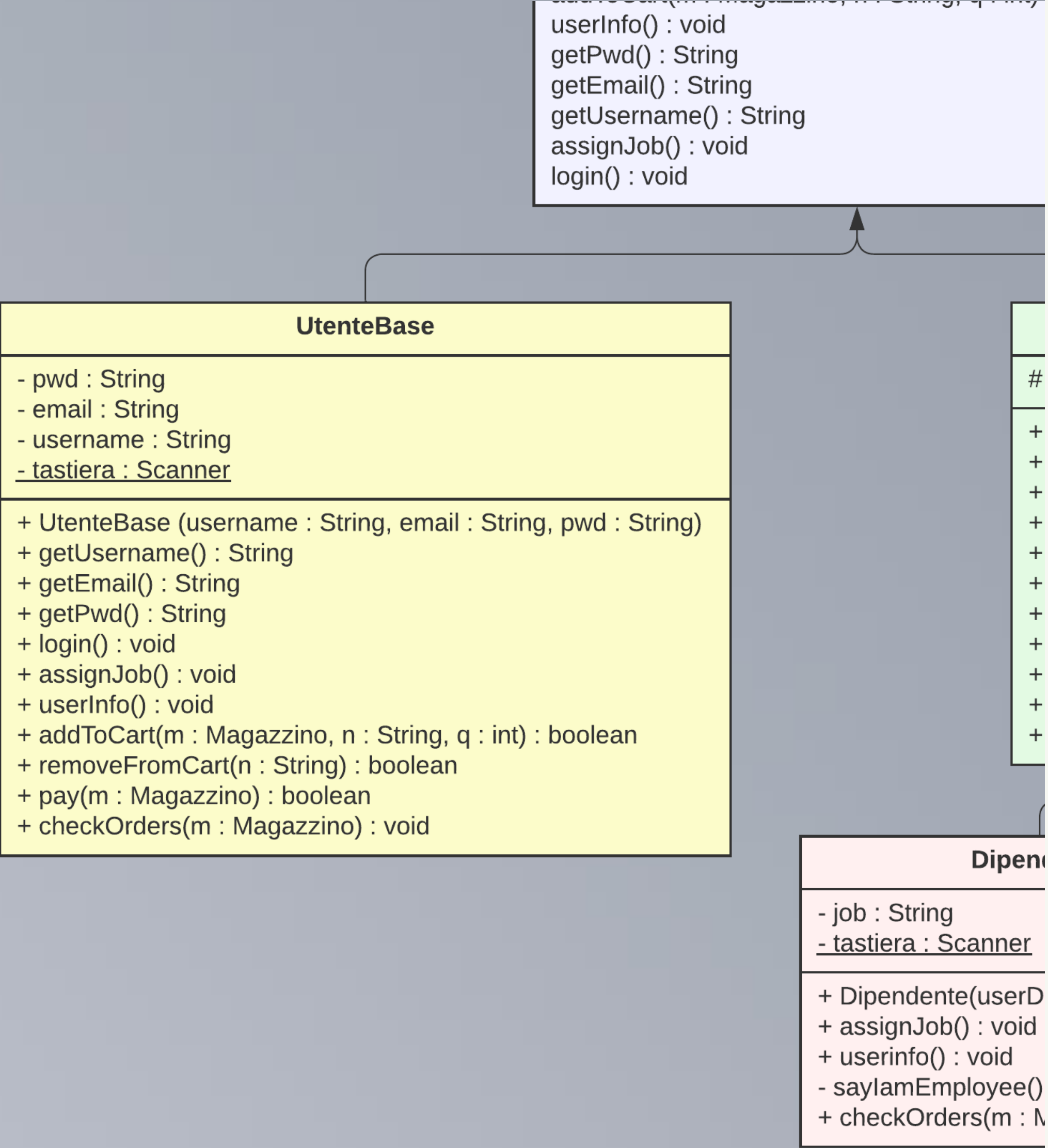
È importante mantenere l'interfaccia User “**leggera**”: il suo scopo è la definizione di una interfaccia conforme, non deve contenere altri dati.

Mettere troppe funzionalità nella classe Component aumenta la possibilità che le sottoclassi concrete “**paghino**” per caratteristiche di cui non hanno un effettivo bisogno.

```
4
5 public interface User {
6
7     public void login();
8
9     public void assignJob();
10
11     public String getUsername();
12
13     public String getEmail();
14
15     public String getPwd();
16
17     public void userInfo();
18
19     public boolean addToChart(Magazzino m, String n, int q);
20
21     public boolean removeFromChart(String n);
22
23     public boolean pay(Magazzino m);
24
25     public void checkOrders(Magazzino m);
26 }
```


ConcreteComponent

Definisce l’oggetto su cui verranno
aggiunte dinamicamente
responsabilità



Decorator

Mantiene una referenza al Componenti
implementando una interfaccia
conforme al Component



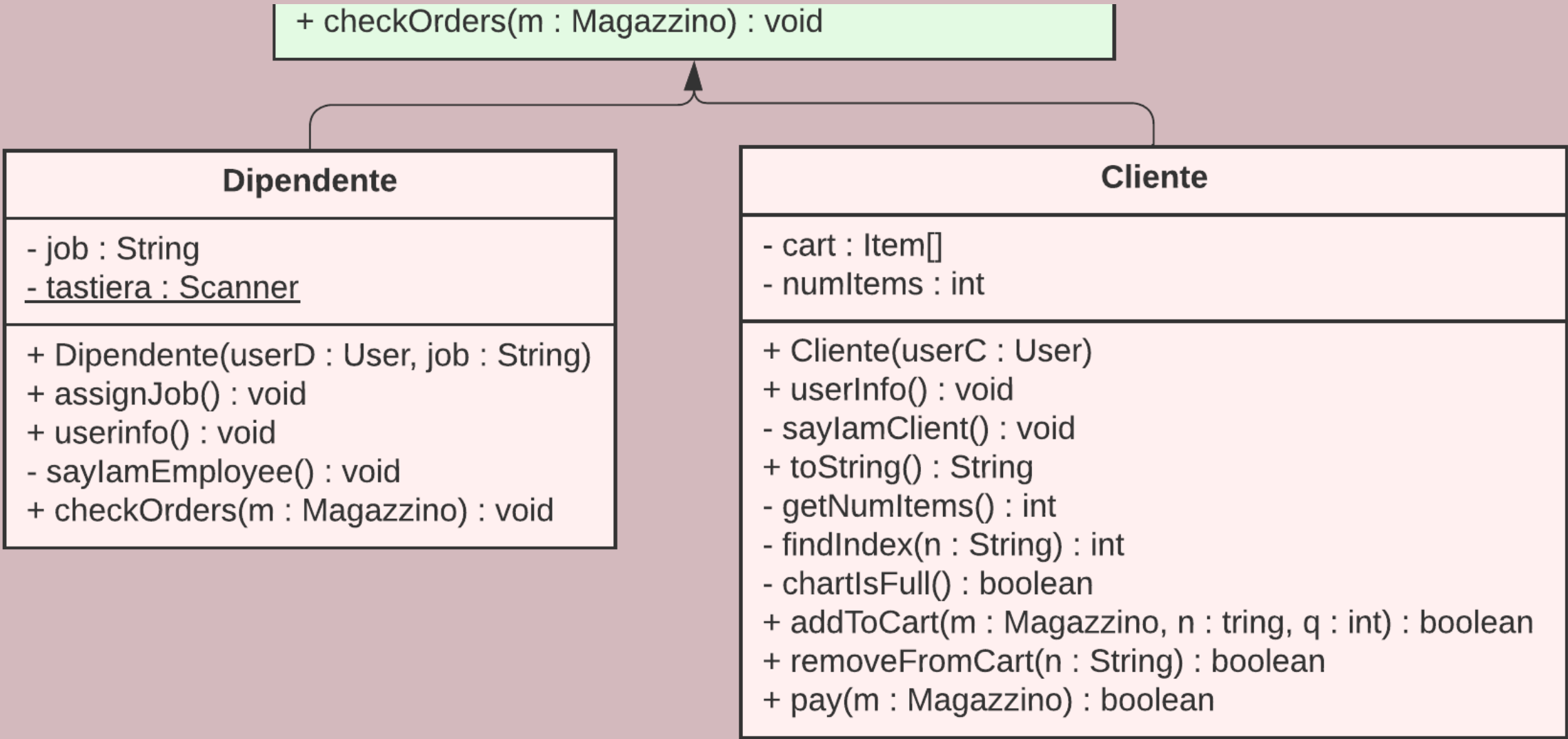
Contiene il codice necessario per immagazzinare al suo interno l'oggetto decorato User, mappa verso di lui le operazioni richieste. Implementa l'interfaccia User, che utilizza per **comunicare** con il Component.

Notare il ruolo chiave del **binding dinamico** che si ottiene ridefinendo i metodi applicati all'oggetto User.

```
5 public abstract class Ruolo implements User {
6
7     protected User appUser;
8
9     public Ruolo(User user) {
10         appUser = user;
11     }
12
13     public void login() {
14     }
15
16     public void assignJob() {
17     }
18
19     public String getUsername() {
20         return appUser.getUsername();
21     }
22
23     public String getEmail() {
24         return appUser.getEmail();
25     }
26
27     public String getPwd() {
28         return appUser.getPwd();
29     }
30
31     public void userInfo() {
32         appUser.userInfo();
33     }
34
35     public boolean addToChart(Magazzino m, String n, int q) {
36         return appUser.addToChart(m, n, q);
37     }
38
39     public boolean removeFromChart(String n) {
40         return appUser.removeFromChart(n);
41     }
42 }
```

ConcreteDecorator

Aggiunge le responsabilità al Component



Le responsabilità specifiche dell'utente **dipendente** sono codificate nella classe Dipendente.

Le responsabilità specifiche dell'utente **cliente** sono codificate nella classe Cliente.

```
40     public void checkOrders(Magazzino m) {
41         if (m.hasOrders()) {
42             System.out.println(m.getOrders());
43         } else {
44             System.out.println("Nessun ordine presente.");
45         }
46     }
47 }
```

```
99     public boolean pay(Magazzino m) {
100         if (numItems == 0) {
101             System.out.println("Cestino vuoto, pagamento non andato a
buon fine.");
102             return false;
103         } else {
104
105             int i = 0;
106             int bill = 0;
107             while (i < numItems) {
108                 m.addOrder(this.getEmail(), cart[i].getName(), cart[i].
getPrice(), cart[i].getQuantity());
109                 bill += cart[i].getQuantity() * cart[i].getPrice();
110                 ++i;
111             }
112
113             //System.out.println("Prodotti acquistati:\n" + this + "
Saldo totale pagato: " + bill + "$");
114             System.out.println("Saldo totale pagato: " + bill + "$");
115
116             this.numItems = 0;
117             this.cart = new Item[10];
118             return true;
119         }
120     }
121 }
```


Problema del Diamante

```
53     User utente2 = new UtenteBase("chiara.genovese", "chiara.  
genovese@me.it", "querty99");  
54     //chiara.userInfo();  
55  
56     User chiara = new Dipendente(utente2, "sarta");  
57     //chiara.assignJob();  
58     System.out.println("\nInformazioni sull'utente dipendente:");  
59     chiara.userInfo();  
60  
61     System.out.println("\nL'utente dipendente Chiara controlla gli  
ordini da evadere.");  
62     chiara.checkOrders(magazzinoSplendor);  
63  
64  
65     // L'utente dipendente Chiara diventa anche un cliente dell'  
Atelier Splendor  
66  
67     System.out.println("\nL'utente dipendente Chiara diventa anche  
un cliente dell'Atelier Splendor.");  
68     User chiara_cliente = new Cliente(chiara);  
69     System.out.println("Informazioni sull'utente dipendente-  
cliente:");  
70     chiara_cliente.userInfo();  
71  
72     // Chiara aggiunge i seguenti items al suo cestino  
73     chiara_cliente.addToChart(magazzinoSplendor, "t-shirt", 1);  
74     chiara_cliente.addToChart(magazzinoSplendor, "giacca", 2);  
75  
76     System.out.println("\nL'utente cliente-dipendente Chiara  
aggiunge i seguenti items al carrello:");  
77     System.out.println(chiara_cliente);  
78  
79     System.out.println("L'utente cliente-dipendente Chiara  
effettua l'ordine ed lei stessa che controlla gli ordini da  
evadere.");  
80     chiara_cliente.pay(magazzinoSplendor);  
81     chiara_cliente.checkOrders(magazzinoSplendor);  
82 }
```

Informazioni sull'utente dipendente:

Sono un utente dipendente con la mansione di sarta. Username: chiara.genovese, Email: chiara.genovese@me.it

L'utente dipendente Chiara controlla gli ordini da evadere.

Ordini in attesa di essere evasi:

cliente: marco.rossi@me.it, item: t-shirt, price: 33\$, quantity: 3

cliente: marco.rossi@me.it, item: giacca, price: 256\$, quantity: 1

cliente: marco.rossi@me.it, item: jeans, price: 80\$, quantity: 1

cliente: marco.rossi@me.it, item: camicia, price: 90\$, quantity: 2

Numero totale di items da spedire: 7

L'utente dipendente Chiara diventa anche un cliente dell'Atelier Splendor.

Informazioni sull'utente dipendente-cliente:

Sono un utente cliente. Sono un utente dipendente con la mansione di sarta. Username: chiara.genovese, Email: chiara.genovese@me.it

Numero di items nel carrello: 0

L'utente cliente-dipendente Chiara aggiunge i seguenti items al carrello:

Carrello:

item: t-shirt, price: 33\$, quantity: 1

item: giacca, price: 256\$, quantity: 2

Numero totale di items: 3

L'utente cliente-dipendente Chiara effettua l'ordine ed è lei stessa che controlla gli ordini da evadere.

Saldo totale pagato: 545\$

Ordini in attesa di essere evasi:

cliente: marco.rossi@me.it, item: t-shirt, price: 33\$, quantity: 3

cliente: marco.rossi@me.it, item: giacca, price: 256\$, quantity: 1

cliente: marco.rossi@me.it, item: jeans, price: 80\$, quantity: 1

cliente: marco.rossi@me.it, item: camicia, price: 90\$, quantity: 2

cliente: chiara.genovese@me.it, item: t-shirt, price: 33\$, quantity: 1

cliente: chiara.genovese@me.it, item: giacca, price: 256\$, quantity: 2

Numero totale di items da spedire: 10

Decorator vs Strategy

Il pattern Decorator e il pattern Strategy costituiscono due modi **alternativi** per cambiare il comportamento di un oggetto.

- Possiamo pensare al Decorator come a una **pelle** che ricopre l'oggetto e che ne cambia il comportamento.
- Quando la classe Component è “**pesante**” (ricca di contenuti) il pattern Decorator è troppo costoso da applicare iterativamente e si preferisce lo strategy.
- Per un cambiamento più **viscerale**, serve invece lo Strategy.
- La differenza principale rispetto al decorator è che ci permette di alterare/estendere le responsabilità andando a **sostituire** l'oggetto Strategy.