

# UNIVERSITA' DEGLI STUDI DI NAPOLI "PARTHENOPE"

DIPARTIMENTO DI SCIENZE E TECNOLOGIE

CORSO DI LAUREA IN INFORMATICA



Relazione Progetto Programmazione III

## **LOGISTICA**

Docente

Angelo Ciaramella

Candidato

Lugubre Domenico

MAT.0124001964

Anno Accademico 2022/2023

## Introduzione

Questo progetto mira a sviluppare un'applicazione dedicata al campo della logistica. L'obiettivo principale è quello di offrire un sistema efficiente per gestire la consegna delle merci, avvalendosi della collaborazione di diverse aziende di trasporto, o corrieri.

Ogni azienda di trasporto, o corriere, possiede una serie di veicoli, ciascuno identificato da un codice univoco, tipo di veicolo, e capacità del container. Quest'ultimo dettaglio è fondamentale, in quanto determina il numero di colli, o pacchi, che il veicolo può trasportare. Ogni collo, a sua volta, è identificato da un insieme di informazioni, tra cui un codice univoco, il mittente, il destinatario e il suo peso.

L'applicazione, oltre a gestire queste informazioni di base, implementa anche un algoritmo approssimato, noto come "Next Fit", per risolvere il problema del Bin Packing. Questo algoritmo garantisce i veicoli siano caricati in modo ottimale, minimizzando gli spazi inutilizzati e massimizzando la capacità di carico.

Ulteriormente, il sistema permette ai corrieri di aggiornare lo stato dei colli ad ogni centro di smistamento. Questa funzionalità è fondamentale, poiché permette ai destinatari di rintracciare i loro pacchi in tempo reale, utilizzando un codice di spedizione univoco.

Nel corso di questa relazione, verranno esplorate in dettaglio le diverse componenti del progetto, mettendo in evidenza come esso risponda alle specifiche esigenze delineate nella traccia e come adotti vari design pattern, tra cui mediator, chain of responsibility, state e singleton, per garantire al sistema robustezza, scalabilità e funzionalità ottimali.

# CAPITOLO 1

## Requisiti

I requisiti del progetto rappresentano le esigenze e le specifiche che il sistema di logistica deve soddisfare.

### Requisiti Funzionali

#### 1) Gestione dei Corrieri:

- Registrazione di un nuovo corriere nel sistema, identificato da un nome (es. "GLS", "DHL").
- Ogni corriere può avere uno o più veicoli associati.
- Caricamento dei colli nei veicoli in base alla disponibilità di spazio, utilizzando l'algoritmo next fit.
- Aggiornamento dello stato dei colli da parte del corriere.

#### 2) Gestione dei Veicoli:

- Ogni veicolo è identificato da un codice univoco, tipo (es. Camion, furgone, ecc.), e capacità del container.
- Possibilità per un veicolo di determinare se un collo specifico può essere inserito al suo interno, basandosi sulla capacità corrente all'inizio uguale a zero e sulla capacità massima del container.
- Aggiunta di un nuovo veicolo ad un corriere, in pratica ogni veicolo sarà associato ad uno specifico corriere.

#### 3) Gestione dei colli:

- Ogni collo è identificato da un codice univoco, mittente, destinatario, peso e codice di spedizione.
- I colli hanno stati definiti (es. Ritiro, InTransito, ec.) e hanno la possibilità di passare dinamicamente da uno stato all'altro.

#### 4) Gestione dei destinatari:

- Ogni destinatario è identificato da un nome, cognome, indirizzo, e-mail e password.
- Il destinatario può avere uno o più colli associati.
- Il destinatario ha la possibilità di tracciare lo stato dei colli associati attraverso l'opportuno codice di spedizione.

#### 5) Interazione con Il Centro di Smistamento:

- Il centro di smistamento funge da mediatore tra i corrieri e i destinatari.
- Fornisce un punto di accesso centralizzato per ottenere informazioni sullo stato di spedizione dei colli.

#### 6) Caricamento dei dati da file:

- Il sistema può caricare dati da un file esterno.
- Il file fornisce informazioni sui corrieri, veicoli, colli, e destinatari.

#### 7) Gestione dell'interfaccia utente:

- Schermata di benvenuto all'avvio dell'applicazione.
- Schermata di login per autenticare i destinatari.
- Una volta autenticati i destinatari possono tracciare i loro colli.

## Requisiti Non Funzionali

### 1) Usabilità:

- Il sistema fornisce un'interfaccia grafica intuitiva, rendendo facile per gli utenti tracciare i loro colli.

### 2) Efficienza:

- L'uso dell'algoritmo Next-Fit garantisce un caricamento efficiente dei colli nei veicoli.

### 3) Sicurezza:

- Solo i destinatari con credenziali valide possono gestire i loro colli.

### 4) Modularità e Estensibilità:

- L'uso di design pattern come mediator, chain of responsibility, state e singleton rende il sistema modulare e facilmente estensibile per future funzionalità.

## Architettura Generale

Il sistema di spedizione è stato progettato per gestire un servizio di tracciamento e gestione delle spedizioni. L'architettura è composta da diverse entità chiave che collaborano tra di loro per assicurare un funzionamento efficiente del sistema.

### Centro di Smistamento

È il fulcro del sistema, agisce come un mediatore, facilitando la comunicazione tra i vari corrieri e destinatari. Il centro utilizza il pattern mediator per gestire le interazioni tra le diverse parti. In particolare, riduce la dipendenza tra i corrieri e i destinatari.

### Corriere

Gestiti tramite una Map all'interno della classe FileManager. Ogni corriere ha una lista di veicoli e colli associati. Implementa la logica di carico dei colli nei veicoli sfruttando le caratteristiche dell'algoritmo Next-fit, garantendo una gestione ottimale dello spazio disponibile nei veicoli. Oltre a ciò, i corrieri sono responsabili dell'aggiornamento dello stato dei colli durante le diverse fasi della spedizione, e questo è stato implementato attraverso l'utilizzo dello ScheduledExecutorService che permette di pianificare l'esecuzione di un task (compito) dopo un certo periodo di tempo, oppure di eseguirlo periodicamente. Fa parte del framework di concorrenza di Java e permette di gestire più thread in modo efficiente.

### Veicoli

Ogni veicolo è associato a un corriere e ha un tipo, una capacità di carico e un codice univoco. I veicoli determinano se un collo può essere inserito al loro interno attraverso il metodo "puoEssereInserito(...)". In particolare, questo metodo verifica se c'è abbastanza spazio nel veicolo per inserire il collo fornito.

### Colli

Ogni collo ha informazioni come mittente, destinatario, peso, un codice univoco, e un codice di spedizione. In particolare, i colli hanno la capacità di cambiare il proprio stato, questo rappresenta la fase attuale di spedizione, cioè a quale punto del processo si trova il collo, e questo viene fatto sfruttando l'utilizzo del pattern State.

### Destinatari

I destinatari sono le entità che ricevono i pacchi. Ogni destinatario ha dettagli personali come nome, cognome, indirizzo, e-mail e password. Posso tracciare i loro colli associati attraverso il corrispettivo codice di spedizione e ricevere aggiornamenti sullo stato dei pacchi.

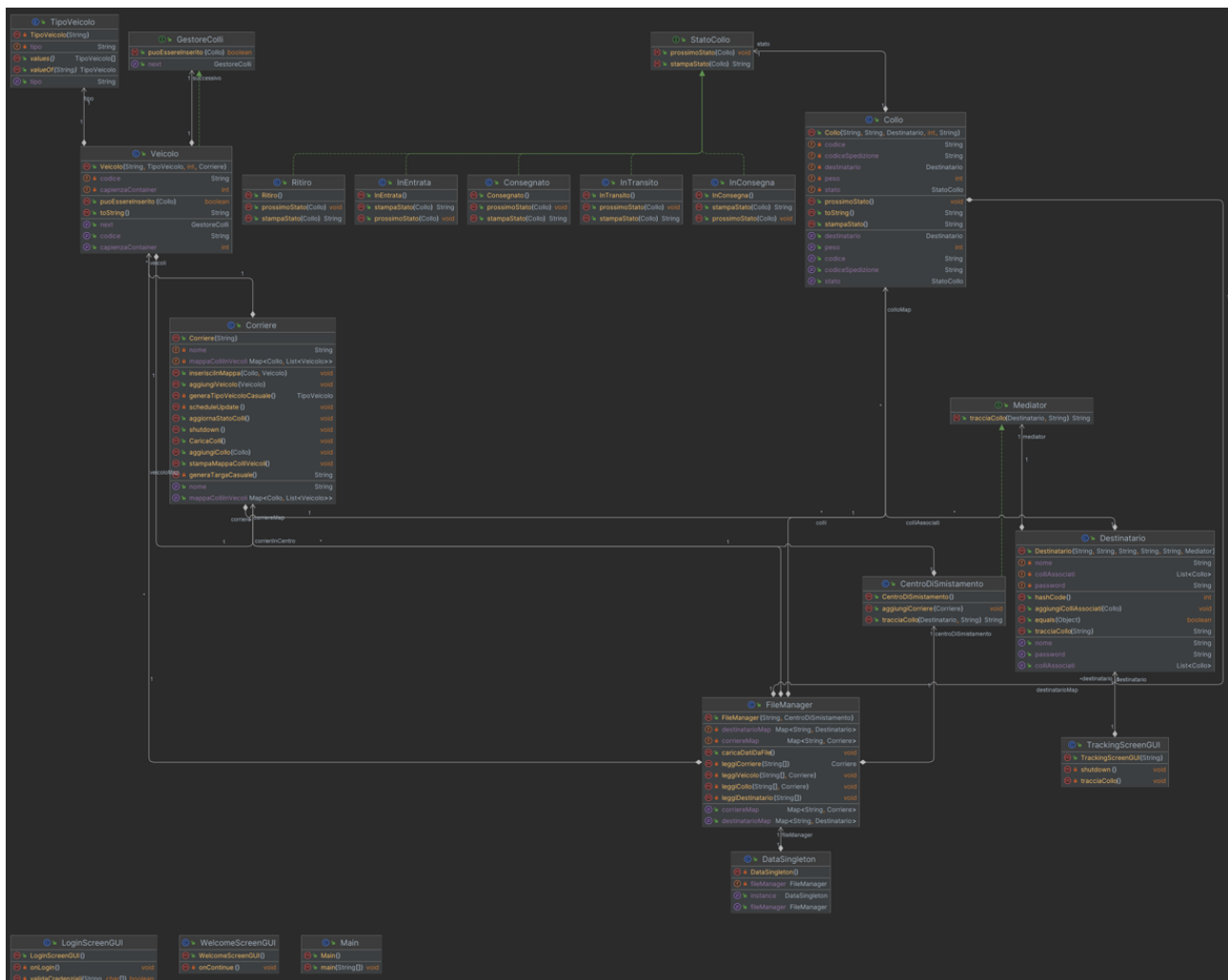
### FileManager

Questa classe è responsabile della persistenza dei dati. Carica le informazioni dal file di configurazione e popola le varie mappe (corriere, colli, destinatari, veicoli) rendendole disponibili nel sistema. Agisce come un singleton attraverso la classe DataSingleton, assicurando che ci sia una sola istanza del FileManager all'interno di tutto il sistema.

### Interfaccia Grafica (GUI)

Il sistema presenta le seguenti interfacce grafiche: WelcomeScreenGUI, LoginScreenGUI e TrackingScreenGUI, le quali forniscono un'interfaccia intuitiva ai destinatari, i quali potranno accedere al sistema, visualizzare informazioni relative ai colli associati al destinatario che ha effettuato l'accesso e permettere il tracking di tali colli facendo inserire al destinatario il codice di spedizione associato ad ogni collo.

# UML GENERALE



## Analisi Diagramma UML:

Classi Concrete Implementate:

- CentroDiSmistamento
- Collo
- Consegnato
- Corriere
- DataSingleton
- Destinataro
- FileManager
- InConsegna
- InEntrata
- InTransito
- LoginScreenGUI
- Main
- Ritiro
- TrackingScreenGUI
- Veicolo
- WelcomeScreenGUI

Interfacce:

- GestoreColli
- Mediator
- StatoCollo

Enumerazioni:

- TipoVeicolo

**Spiegazione Relazioni:**

1) Associazioni:

- **FileManager-CentroDiSmistamento:** il FileManager ha un riferimento al CentroDiSmistamento.
- **Destinatario-Mediator:** Il Destinatario comunica con il Mediator per ottenere informazioni sullo stato della spedizione.
- **Veicolo-GestoreColli:** è un'associazione perché un veicolo può esistere anche se non fa parte di una catena di responsabilità.

2) Aggregazioni:

- **CentroDiSmistamento-Corriere:** Il CentroDiSmistamento gestisce diversi Corrieri, ma ogni corriere può esistere indipendentemente dal CentroDiSmistamento e viceversa. Il centro ha una lista di corrieri associati ma questa all'inizio potrebbe essere vuota, e, in seguito, popolata con vari corrieri.
- **Corriere-Collo:** anche questa può essere vista come un'aggregazione. Un corriere gestisce molti colli, ma il ciclo di vita dell'oggetto parte in questo caso collo non dipende dal ciclo di vita dell'oggetto contenitore in questo caso corriere, perché un collo potrebbe essere semplicemente non essere stato assegnato ad un corriere.

3) Composizioni:

- **FileManager (Corriere, Collo, Veicolo, Destinatario):** Il FileManager gestisce le mappe di queste entità. Se il FileManager venisse distrutto queste mappe andrebbero perse, quindi è una relazione forte di composizione.
- **Destinatario-Collo:** Un Destinatario può avere più Colli, ma un Collo non può esistere senza un destinatario.
- **Collo-StatoCollo:** Il Ciclo di vita dello stato del collo è strettamente legato al ciclo di vita dell'oggetto collo. In particolare, quando un Collo viene distrutto anche il suo stato associato viene distrutto. Inoltre, lo stato non ha senso senza un Collo a cui è associato. (ok)
- **Corriere-Veicolo:** Il corriere ha diversi veicoli, se il corriere viene distrutto anche i veicoli associati ad esso vengono distrutti.

## CAPITOLO 2

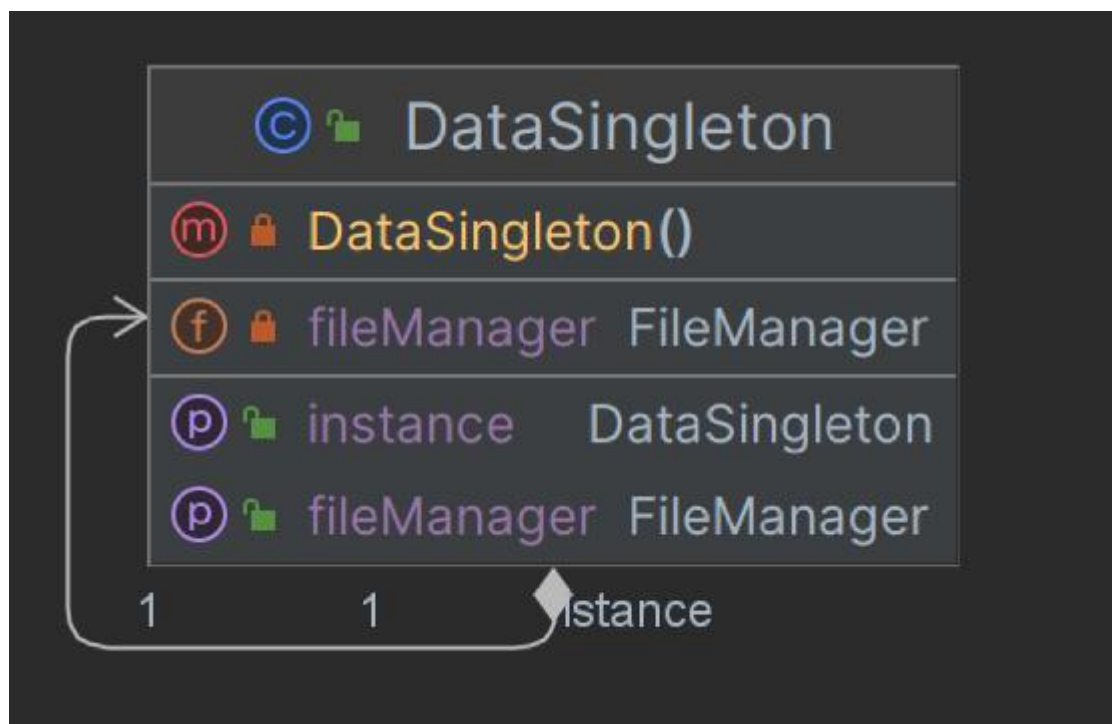
### Design Pattern

#### Singleton

Descrizione:

Il pattern Singleton assicura che una classe abbia una sola istanza e fornisce un punto di accesso globale a questa istanza. Questo pattern può essere particolarmente utile quando un'unica istanza centralizzata può controllare l'accesso alle risorse, garantendo che non ci siano conflitti o duplicazioni indesiderate.

Applicazione nel Progetto:



Nel progetto, il Singleton è utilizzato attraverso la classe `DataSingleton`. Questa classe ha il compito di inizializzare e gestire l'accesso centralizzato ai dati attraverso il `FileManager`. L'obiettivo è garantire che, in tutto il sistema ci sia un unico `FileManager` che gestisca prima la lettura e, successivamente la gestione dei dati. La decisione di utilizzare il pattern Singleton per la classe `DataSingleton` assicura che, indipendentemente da dove venga richiamata nel programma, l'istanza restituita sia sempre la stessa. Questo previene potenziali conflitti o problemi di sincronizzazione ai dati se diverse parti del programma tentassero di inizializzare o di accedere a istanze diverse del `FileManager`.

Per implementare il singleton la classe `DataSingleton` ha un costruttore privato, che impedisce la creazione di nuove istanze dall'esterno. Inoltre, mantiene un riferimento statico alla sua unica istanza attraverso la variabile `instance`. La funzione `getInstance()` si occupa di restituire quest'unica istanza, creandone una nuova se non esiste già.

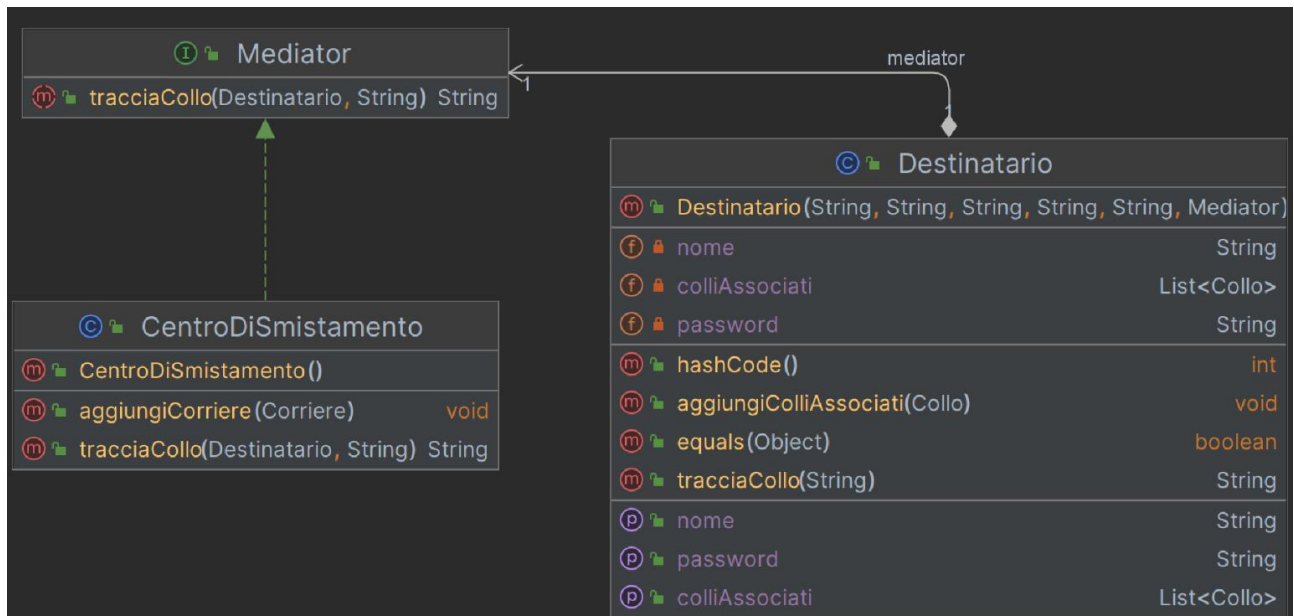


## Mediator

Descrizione:

Il pattern Mediator fornisce un meccanismo per ridurre le complessità delle comunicazioni tra le varie classi. In particolare, definisce un oggetto che incapsula il meccanismo di interazione di oggetti, quindi consentendo il loro disaccoppiamento in modo da variare facilmente le interazioni tra di loro. Quindi gli oggetti non comunicano direttamente tra di loro, ma interagiscono solo attraverso il mediatore.

Applicazione nel Progetto:



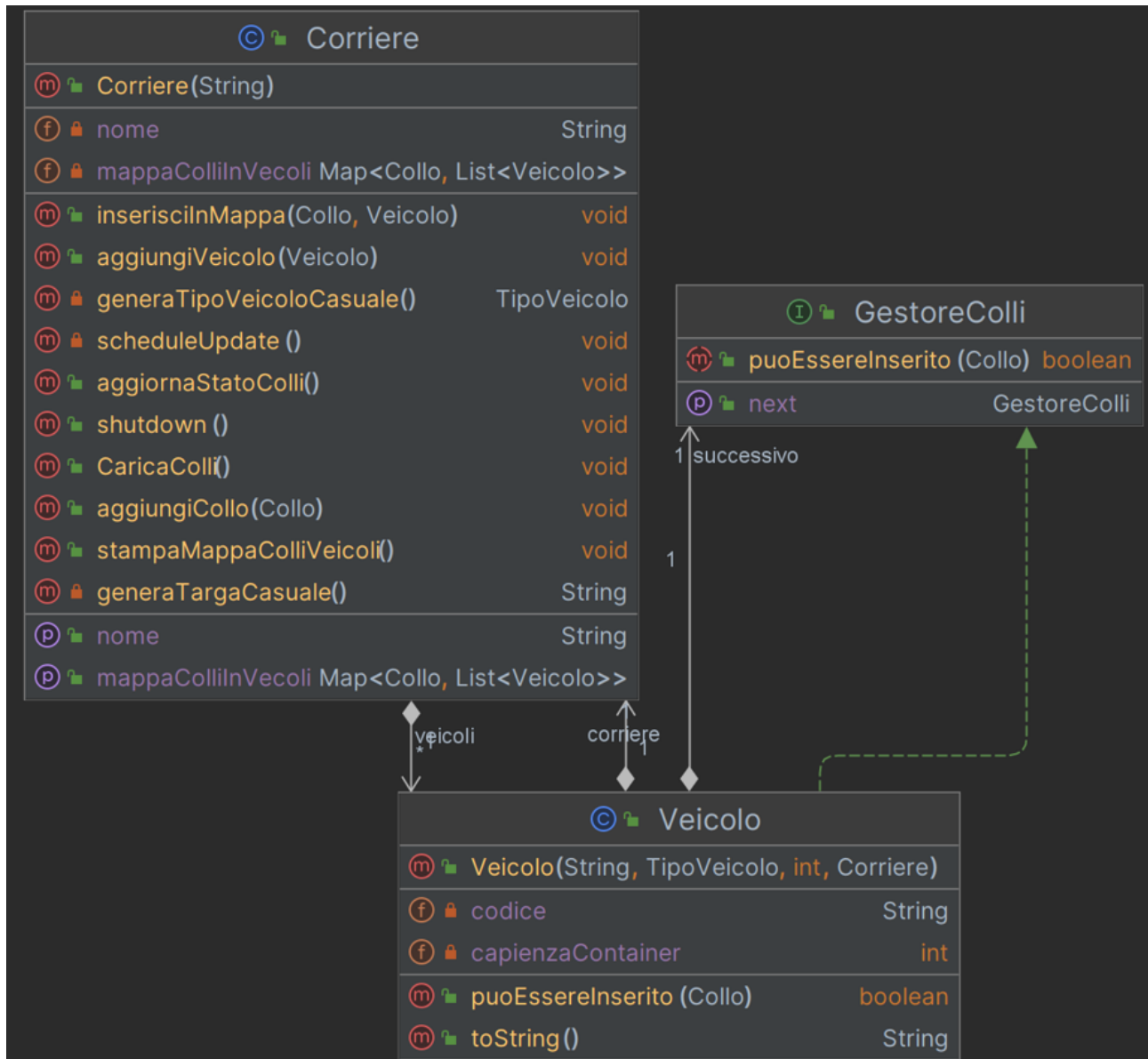
Nel progetto, il centro di smistamento agisce da mediatore. Il suo ruolo è quello di gestire e coordinare le interazioni tra la classe Corriere e la classe Destinatario, in modo tale che queste due classi non debbano comunicare direttamente tra di loro. Questa scelta di design aiuta a mantenere le separazioni delle responsabilità e rende il sistema più flessibile e manutenibile. Ad esempio, quando un Destinatario vuole tracciare un collo, non interroga direttamente un corriere. Invece il destinatario richiama il metodo `tracciaCollo` sul mediatore (`CentrodiSmistamento`), che poi si occupa di trovare le informazioni necessarie attraverso il corriere appropriato e le restituisce al destinatario. Questa implementazione del pattern mediator centralizza la logica di interazione tra le varie entità, rendendo il sistema più ordinato e riduce le dipendenze incrociate tra le classi.

## Chain of Responsibility

Descrizione:

Questo design pattern permette di passare le richieste lungo una catena di potenziali gestori fino a quando uno di essi la gestisce o fino a quando la catena è terminata. Questo modello consente di separare il mittente di una richiesta dal destinatario, dando la possibilità a più di un oggetto (destinatario) di gestire la richiesta.

Applicazione nel Progetto:



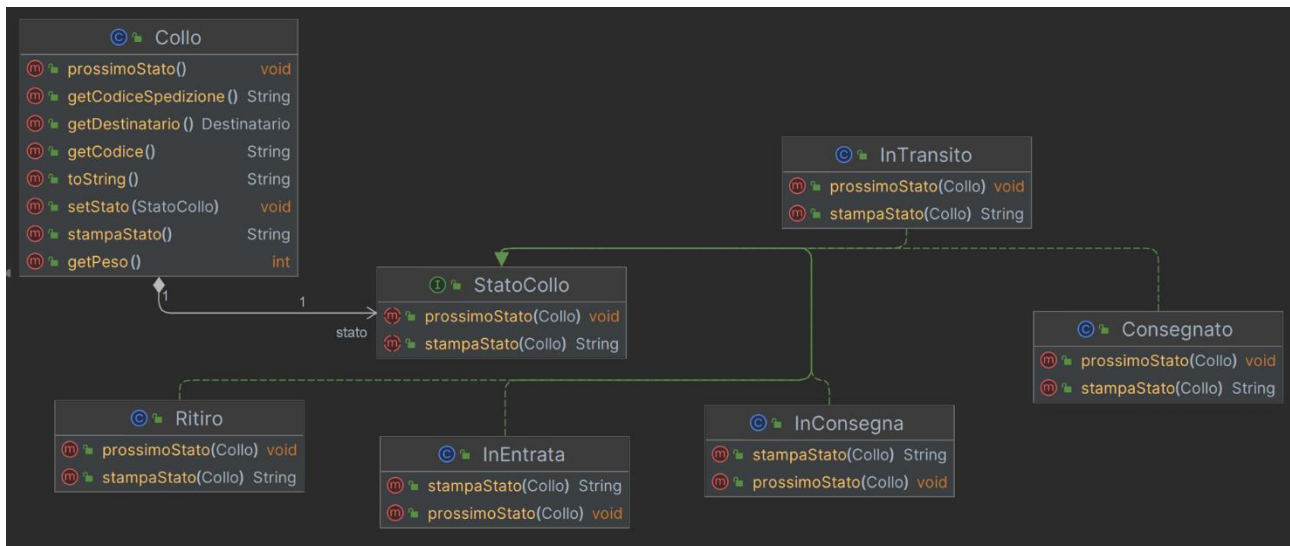
All'interno del progetto, il pattern CoF è stato utilizzato in combinazione con l'algoritmo NextFit per gestire il carico dei colli nei veicoli. Ogni veicolo che implementa l'interfaccia **GestoreColli**, rappresenta un gestore nella catena. Quando un collo deve essere caricato, allora viene utilizzato l'algoritmo next fit: il collo viene passato al primo veicolo disponibile nella catena e, se quel veicolo ha abbastanza spazio, il collo viene caricato. Se non c'è spazio sufficiente, il collo viene passato al prossimo veicolo nella catena, e così via, fino a trovare un veicolo che può caricarlo. La classe **Corriere** costruisce questa catena di veicoli, in particolare ogni veicolo che implementa l'interfaccia **GestoreColli**, ha un riferimento al suo successore nella catena (il prossimo veicolo a cui passare il collo se non può essere gestito). La catena viene creata nel metodo `aggiungiVeicolo`, mentre la logica del NextFit è implementata nel metodo `caricaColli` nella classe **Corriere**.

# State

Descrizione:

il pattern State permette ad un oggetto di cambiare il suo comportamento quando il suo stato interno cambia, è come se l'oggetto cambiasse di classe. Questo design pattern è particolarmente utile quando un oggetto deve eseguire comportamenti differenti a seconda dello stato in cui si trova e quando questi stati e transizioni sono precisamente definite.

Applicazione nel Progetto:



All'interno del progetto il pattern State è utilizzato per gestire gli stati dei Colli, poiché un collo può attraversare diversi stati durante il suo ciclo di vita come ritiro, in transito e consegnato. Ogni stato è rappresentato da una classe separata, e il collo tiene un riferimento al suo stato corrente. Questo permette al collo di comportarsi in modo diverso a seconda dello stato in cui si trova. Se in futuro si desidera aggiungere nuovi stati, il sistema può facilmente essere modificato grazie all'uso del pattern State, il quale ci ha fornito una struttura più modularizzata e flessibile.

# CAPITOLO 3

## Dettagli implementativi

Di seguito sono riportate alcune porzioni rilevanti del codice sviluppato.

### Creazione della catena di responsabilità

```
/**
 * Aggiunge un veicolo alla lista dei veicoli gestita dal Corriere.
 * Se nella lista sono già presenti dei veicoli allora imposta il nuovo veicolo come successivo dell'ultimo
 * veicolo già presente nella lista, seguendo la logica del pattern chain of responsibility.
 * @param v Veicolo da aggiungere.
 */

1 usage  ▲ mimmolg
public void aggiungiVeicolo(Veicolo v) {
    if (!veicoli.isEmpty()) {
        veicoli.get(veicoli.size() - 1).setNext(v);
    }
    veicoli.add(v);
}
```

### Next Fit

Il metodo `puoEssereInserito()` controlla se un determinato collo può essere inserito in un veicolo basandosi sulla capacità rimanente del container del veicolo. Questo metodo rappresenta una componente fondamentale dell'algoritmo next fit.

```
/**
 * Determina se un dato collo può essere inserito nel veicolo corrente.
 *
 * Questo metodo verifica se c'è abbastanza spazio nel veicolo per inserire il collo fornito.
 * Se il veicolo ha abbastanza spazio, il collo viene inserito e la funzione ritorna true.
 * Se il veicolo non ha spazio sufficiente ed esiste un veicolo successivo nella catena di responsabilità,
 * la richiesta viene passata al veicolo successivo.
 * Se non esistono altri veicoli nella catena e il veicolo corrente non può accettare il collo,
 * la funzione ritorna false.
 *
 * @param collo Il collo da inserire.
 * @return true se il collo può essere inserito nel veicolo corrente o in un veicolo successivo nella catena;
 *         false altrimenti.
 */

2 usages  ▲ mimmolg
@Override
public boolean puoEssereInserito(Collo collo) {
    if (collo.getPeso() + caricoCorrente <= capienzaContainer) {
        caricoCorrente += collo.getPeso();
        corriere.inserisciInMappa(collo, this);
        return true;
    } else if (successivo != null) {
        return successivo.puoEssereInserito(collo);
    } else {
        return false;
    }
}
```

La vera logica del next fit è invece implementata nell'algoritmo caricaColli() all'interno della classe corriere.

```
/**
 * Carica i colli nei veicoli disponibili seguendo una strategia ottimizzata.
 * Questo metodo:
 * Odina la lista dei veicoli e dei colli in maniera decrescente in base alla capienza e al peso, rispettivamente.
 * Prova a inserire ogni collo in un veicolo disponibile utilizzando il pattern chain of responsibility e la tecnica dell'algoritmo next fit.
 * Se un collo non può essere gestito da nessuno dei veicoli esistenti, viene creato un nuovo veicolo.
 * La strategia mira a garantire che i colli siano caricati in modo efficiente.
 */
public void CaricaColli() {
    int indiceCorrente = 0;
    veicoli.sort(Comparator.comparingInt(Veicolo::getCapienzaContainer).reversed()); //ordino in maniera decrescente per ottimizzare
    colli.sort(Comparator.comparingInt(Collo::getPeso).reversed());
    List<Collo> colliNonGestiti = new ArrayList<>();
    //ottengo l'iteratore della collezione
    Iterator<Collo> iteratorColli = colli.iterator();
    while (iteratorColli.hasNext()) { //restituisce true se la lista ha ancora elementi
        Collo collo = iteratorColli.next(); //restituisce il prossimo elemento
        boolean colloInserito = false; //controllo se sono riuscito a gestire o meno il collo
        for (int i = 0; i < veicoli.size(); i++) {
            Veicolo v = veicoli.get((indiceCorrente + i) % veicoli.size()); //ottengo ciclicamente il prossimo indice
            if (v.puoEssereInserito(collo)) {
                colloInserito = true;
                indiceCorrente = (indiceCorrente + i) % veicoli.size();
                break;
            }
        }
        if (colloInserito) {
            iteratorColli.remove(); //il collo è stato inserito e lo rimuoviamo dalla lista
        } else {
            colliNonGestiti.add(collo);
            iteratorColli.remove();
        }
    }
    if (!colliNonGestiti.isEmpty()) {
        for (Collo c : colliNonGestiti) {
            String targaCasuale = this.generaTargaCasuale();
            TipoVeicolo tipoCasuale = this.generaTipoVeicoloCasuale();
            Veicolo nuovoVeicolo = new Veicolo(targaCasuale, tipoCasuale, c.getPeso(), this);
            veicoli.add(nuovoVeicolo);
            this.inserisciInMappa(c, nuovoVeicolo);
        }
    }
}
```

In questo metodo i colli vengono inseriti nei veicoli in un certo ordine finché non si trova un veicolo che può gestirlo. Se non si riesce a trovare un veicolo in grado di gestire il collo allora, viene creato un nuovo veicolo.

## Aggiornamento Automatico Dello Stato dei colli

Per aggiornare lo stato dei colli in maniera automatica ho utilizzato due metodi:

```
/**
 * Pianifica un task periodico per aggiornare lo stato dei colli gestiti dal corriere.
 */
1 usage  mimmolg
private void scheduleUpdate() {
    scheduledExecutorService.scheduleAtFixedRate( command: () -> {
        aggiornaStatoColli();
    }, initialDelay: 20, period: 60, unit: TimeUnit.SECONDS);
}

/**
 * Aggiorna lo stato di ogni collo nella mappa dei colli gestiti dal corriere.
 * Questo metodo sincronizza l'accesso alla mappa dei colli e avanza lo stato di
 * ogni collo alla sua prossima fase, seguendo il pattern State.
 */
2 usages  mimmolg
public void aggiornaStatoColli() {
    synchronized (mappaColliInVecoli) {
        for (Collo c : mappaColliInVecoli.keySet()) {
            c.prossimoStato();
        }
    }
}
```

Il primo è `scheduleUpdate()`. Il metodo utilizza uno “`scheduledExecutorService`”, che è una classe Java che permette di gestire operazioni che devono essere eseguite ad intervalli regolari. La funzione `scheduledAtFixedRate` viene utilizzata per eseguire il metodo `aggiornaStatoColli()` ogni 60 secondi, con un ritardo iniziale di 20 secondi all’avvio dello scheduling. Quindi dopo 20 secondi dall’avvio dell’applicazione, ogni 60 secondi viene aggiornato lo stato del collo cosicché l’utente possa visualizzare le varie fasi del collo ogni volta che prova a tracciare un pacco.

Il metodo `aggiornaStatoColli()` avanza lo stato di ogni collo alla sua prossima fase seguendo il pattern State. Per garantire che l’aggiornamento dello stato dei colli avvenga in modo thread-safe, il metodo usa un blocco `synchronized` che sincronizza l’accesso alla mappa dei colli, `mappaColliInVecoli`.

## Tracking Stato Dei Colli

Quando un destinatario vuole tracciare un pacco richiama il metodo `tracciaCollo(string)` fornendo il codice di spedizione del collo che vuole tracciare.

```
/**
 * Traccia un collo specifico per questo destinatario utilizzando il mediatore associato.
 * @param codice Il codice di spedizione univoco del collo da tracciare.
 * @return Una stringa che descrive lo stato attuale del collo.
 */
1 usage  mimmolg
public String tracciaCollo(String codice) {
    String stato = mediator.tracciaCollo( destinatario: this, codice);
    return stato;
}
```

All'interno del metodo `tracciaCollo` del destinatario, c'è una chiamata al mediatore con la chiamata `mediator.tracciaCollo(this,codice)`. In sostanza, il destinatario si affida al mediatore per recuperare lo stato del collo, e lo fa inviando se stesso (`this`) quindi il destinatario corrente e il codice di spedizione al mediatore. Il metodo `tracciaCollo(this,codice)` viene quindi eseguito. Questo metodo cerca il collo desiderato ricercando tra tutti i corrieri disponibili e i loro colli associati:

```
/**
 * Traccia un collo specifico basandosi sul destinatario e sul codice di spedizione.
 *
 * <p>
 * Questo metodo cerca il collo attraverso tutti i corrieri presenti nel centro di smistamento.
 * Se il collo viene trovato, restituisce lo stato corrente del collo.
 * </p>
 *
 * @param destinatario L'oggetto {@link Destinatario} associato al collo.
 * @param codice Il codice di spedizione univoco del collo.
 * @return Una stringa che rappresenta lo stato del collo o un messaggio di errore se il collo non viene trovato.
 */
1 usage 1 mimmolg
@Override
public String tracciaCollo(Destinatarior destinatario, String codice) {
    for (Corriere corriere : corrieriInCentro.values()) {
        Map<Collo, List<Veicolo>> mappaColliInVecoli = corriere.getMappaColliInVecoli();
        for (Map.Entry<Collo, List<Veicolo>> entry : mappaColliInVecoli.entrySet()) {
            Collo collo = entry.getKey();
            if (collo.getDestinatario().equals(destinatario)) {
                if (collo.getCodiceSpedizione().equals(codice)) {
                    return collo.stampaStato();
                }
            }
        }
    }
    return "Collo non trovato per il destinatario ";
}
```

itera su tutti i corrieri presenti nel centro, una volta selezionato un corriere recupera la mappa dei colli e dei veicoli di tale corriere. Quindi successivamente itera sulla mappa del corriere selezionato e se trova un collo che corrisponde al destinatario fornito e al codice di spedizione, restituisce lo stato del collo. Quindi una volta che il mediatore attraverso questo meccanismo ha trovato lo stato del collo, restituisce lo stato al destinatario. Il metodo `tracciaCollo(String codice)` del destinatario riceve questo stato e lo restituisce come risultato. In sintesi, il destinatario, invece di cercare direttamente tra tutti i colli di tutti i corrieri, delega questa responsabilità al mediatore. Il mediatore agisce come un intermediario, eseguendo la ricerca effettiva e fornendo i risultati al destinatario.

## Interfaccia Utente (GUI)

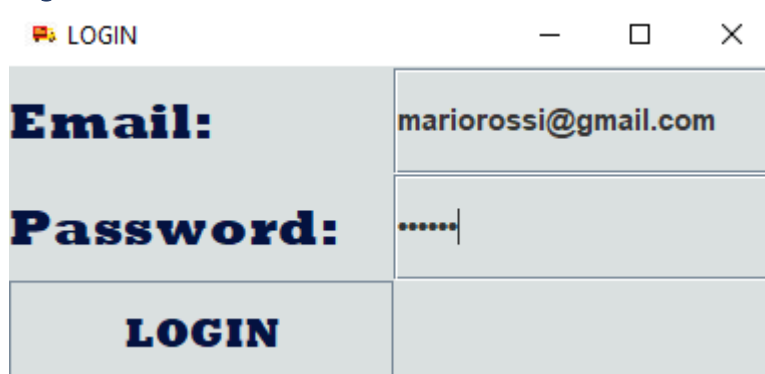
L'applicazione presenta un'interfaccia utente semplice ed intuitiva per monitorare lo stato delle spedizioni. Di seguito sono descritte le varie schermate e le loro funzionalità.

### WelcomeScreenGUI



All'avvio dell'applicazione l'utente è accolto da una schermata di benvenuto. Questa schermata presenta un'immagine di benvenuto e un pulsante per continuare. Una volta cliccato il pulsante l'utente viene indirizzato alla schermata di login.

### LoginScreenGUI



|                  |                      |
|------------------|----------------------|
| <b>Email:</b>    | mariorossi@gmail.com |
| <b>Password:</b> | .....                |
| <b>LOGIN</b>     |                      |

In questa schermata l'utente può inserire le proprie credenziali (e-mail e password). Dopo aver inserito le credenziali l'utente può cliccare sul pulsante login per procedere. Se le credenziali sono corrette l'utente viene indirizzato alla schermata di tracking dei pacchi altrimenti viene mostrato un messaggio di errore e l'utente può reinserire le credenziali corrette.



## TrackingScreenGUI

Tracking Dei Pacchi

**INSERISCI CODICE SPEDIZIONE :**

**TRACCIA SPEDIZIONE**

Collo{codice='GLS03', mittente='Carlo'} ->Codice Spedizione :10936660  
Collo{codice='GLS06', mittente='Marianna'} ->Codice Spedizione :10937454  
Collo{codice='GLS07', mittente='Veronica'} ->Codice Spedizione :10935416  
Collo{codice='GLS08', mittente='Antonio'} ->Codice Spedizione :42389536

In questa schermata l'utente può visualizzare una lista di colli associati al suo account. Vi è un campo dove l'utente può inserire il codice di spedizione per tracciare un collo specifico e al di sotto c'è una text area dove viene mostrato lo stato di questo collo.

Tracking Dei Pacchi

**INSERISCI CODICE SPEDIZIONE :**

**TRACCIA SPEDIZIONE**

Collo{codice='GLS03', mittente='Carlo'} ->Codice Spedizione :10936660  
Collo{codice='GLS06', mittente='Marianna'} ->Codice Spedizione :10937454  
Collo{codice='GLS07', mittente='Veronica'} ->Codice Spedizione :10935416  
Collo{codice='GLS08', mittente='Antonio'} ->Codice Spedizione :42389536

**Il collo GLS03 è stato correttamente consegnato al cliente**

Tracking Dei Pacchi

**INSERISCI CODICE SPEDIZIONE :**

**TRACCIA SPEDIZIONE**

Collo{codice='GLS03', mittente='Carlo'} ->Codice Spedizione :10936660  
Collo{codice='GLS06', mittente='Marianna'} ->Codice Spedizione :10937454  
Collo{codice='GLS07', mittente='Veronica'} ->Codice Spedizione :10935416  
Collo{codice='GLS08', mittente='Antonio'} ->Codice Spedizione :42389536

**Collo non trovato per il destinatario**

Tracking Dei Pacchi

**INSERISCI CODICE SPEDIZIONE :**

**TRACCIA SPEDIZIONE**

Collo{codice='GLS03', mittente='Carlo'} ->Codice Spedizione :10936660  
Collo{codice='GLS06', mittente='Marianna'} ->Codice Spedizione :10937454  
Collo{codice='GLS07', mittente='Veronica'} ->Codice Spedizione :10935416  
Collo{codice='GLS08', mittente='Antonio'} ->Codice Spedizione :42389536

**Il collo GLS06 è in Viaggio verso l'HUB più vicino al punto di consegna**

