DESIGN

Indice

1) Obiettivo del documento

Questo documento ha lo scopo di fornire una descrizione precisa e approfondita della progettazione dell'applicazione **Gestione Torneo Calcio a 5**, includendo il diagramma delle classi, i diagrammi di sequenza ed ulteriori diagrammi, quali il diagramma dei package e il diagramma delle attività**,** necessari a chiarire il funzionamento di alcune parti del software.

2) Diagramma dei package

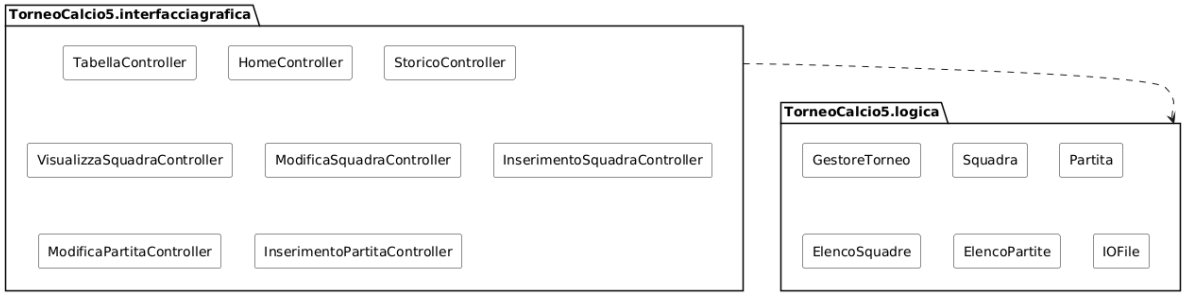
Un diagramma dei package illustra come le classi sono organizzate in package. Il software è diviso in due package :

- **TorneoCalcio5.logica**

Si occupa della logica dell'applicazione, gestisce i dati e include tutte le classi che si occupano delle funzionalità base della gestione del torneo.

- **TorneoCalcio5.interfacciagrafica**

Include tutte le classi che lavorano sull'interfaccia grafica. I file FXML si trovano nelle risorse del progetto e le classi di questo package vi hanno accesso e ne costituiscono la logica applicativa.

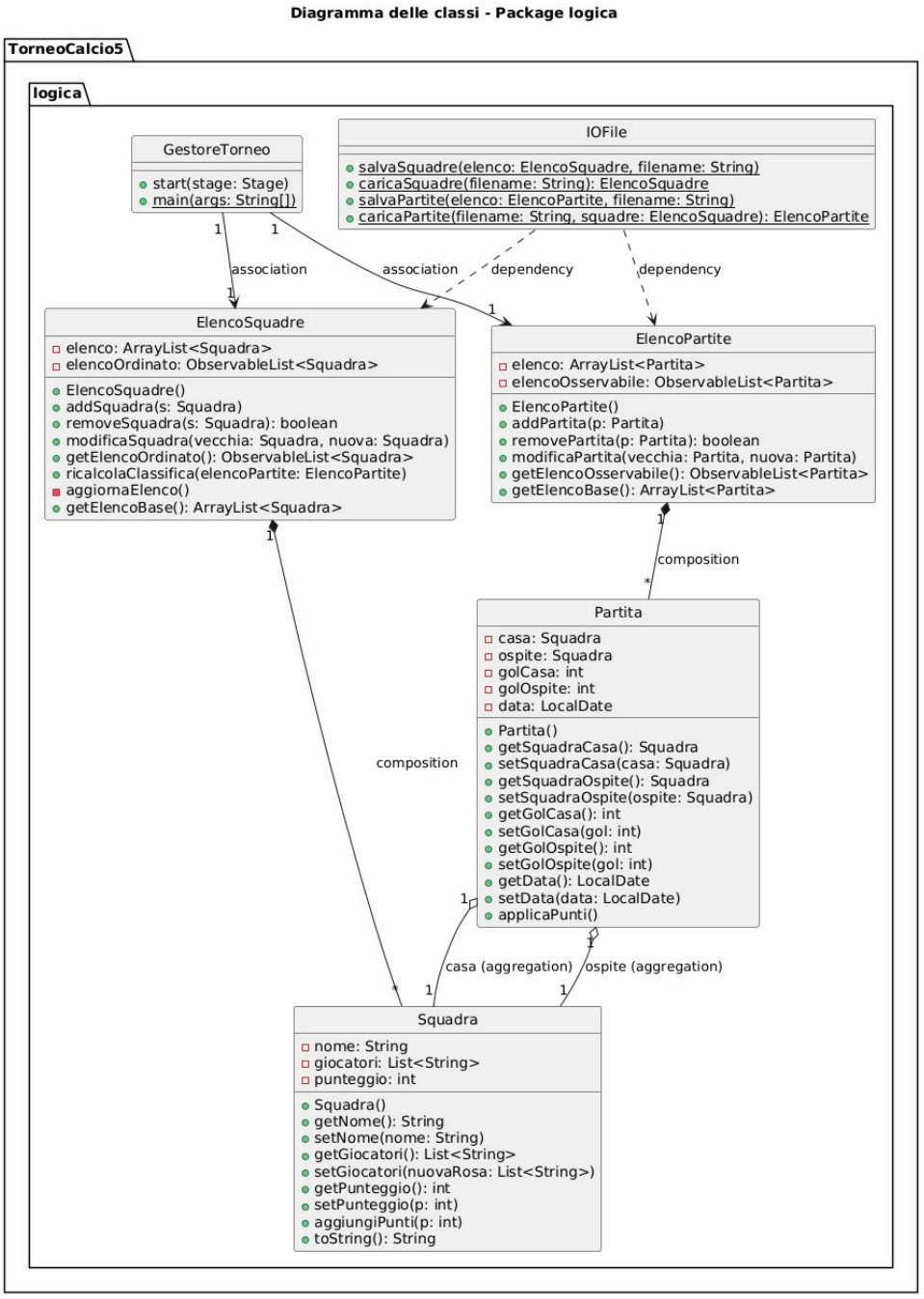


3) Diagramma delle classi

Un diagramma delle classi è un diagramma che permette di descrivere le caratteristiche delle classi e le relazioni tra le stesse. Abbiamo creato due diagrammi dettagliati utilizzando EasyUML per illustrare le dipendenze, le composizioni e le gerarchie di ereditarietà del sistema.

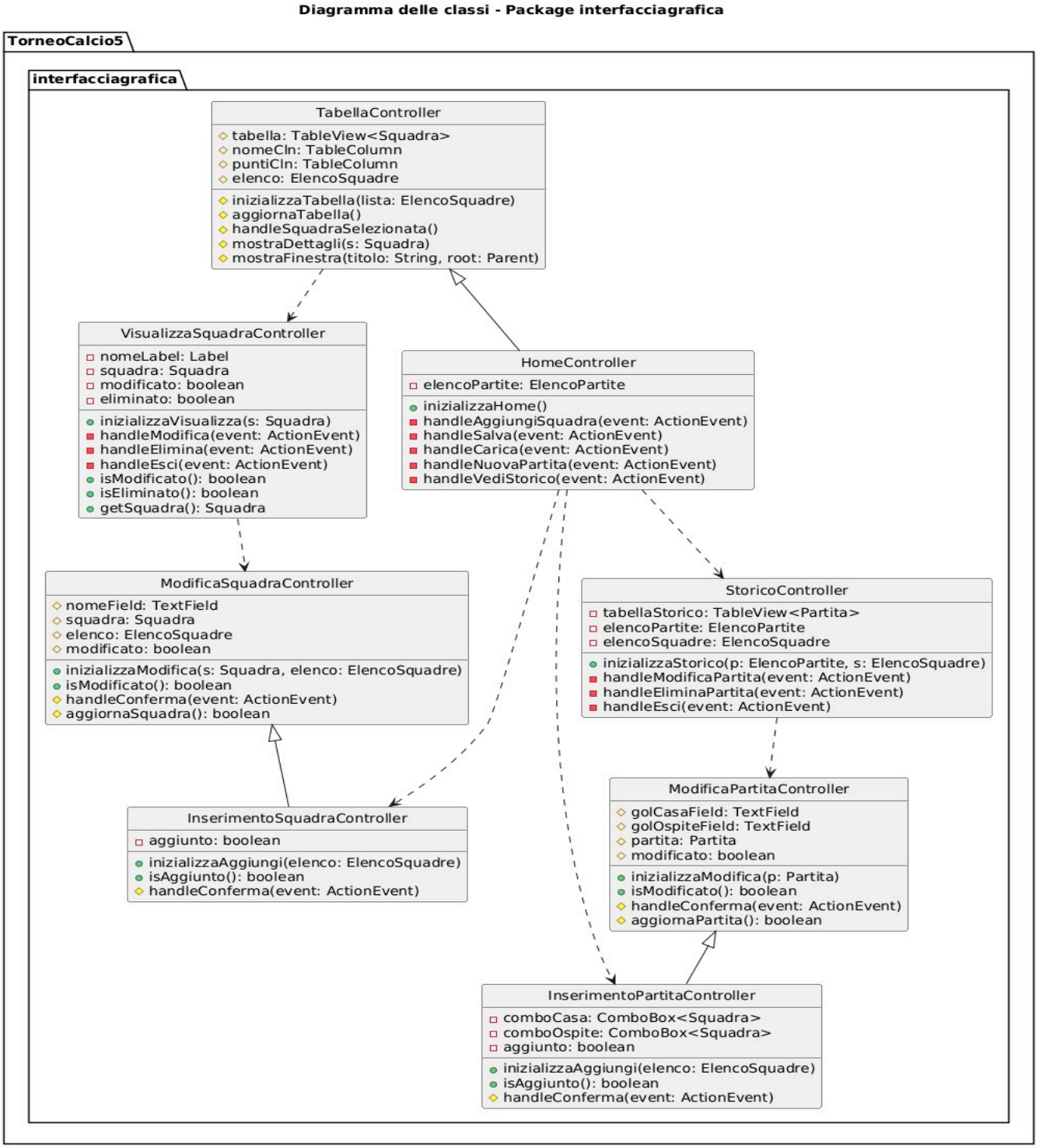
Il package "logica"

Il primo diagramma descrive le relazioni tra le classi del package `logica`, contenente la logica fondamentale dell'applicazione e la gestione dei dati.



Il package "interfacciagrafica"

Il secondo diagramma descrive le relazioni tra le classi del package `interfacciagrafica`, contenente tutti i controller relativi alle diverse finestre dell'interfaccia. Sono omesse le classi della logica per esplicitare meglio le interazioni tra i componenti grafici.



3.1) Coesione e Accoppiamento

L'organizzazione del software segue un'architettura a finestre, dove ogni controller gestisce esclusivamente la propria view (finestra). Questa scelta ci permette di raggiungere una **coesione funzionale** elevata: ogni classe dei controller (come `InserimentoPartitaController` o `VisualizzaSquadraController`) implementa solo la logica necessaria alla gestione della specifica finestra di competenza.

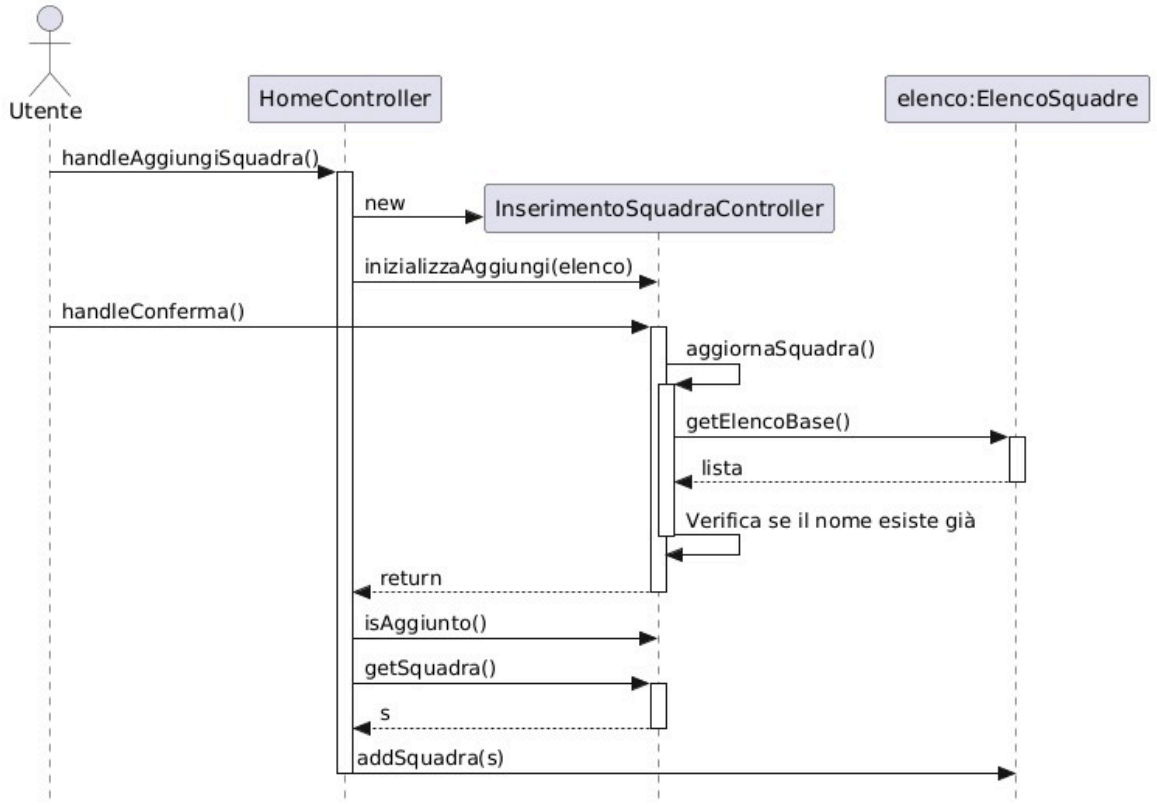
La logica di business (punteggi, classifiche, I/O) è separata nel package `logica`, garantendo che le classi come `Squadra` o `ElencoPartite` assolvano un unico compito ben definito. L'**accoppiamento** raggiunto è di tipo **per dati**. Come si evince dai diagrammi, i moduli scambiano solo le informazioni essenziali (oggetti o flag booleani) tramite parametri, evitando legami rigidi tra le classi.

4) Diagramma di sequenza

Un diagramma di sequenza descrive il comportamento di un sottoinsieme del sistema in uno scenario. Grazie allo stesso è possibile visualizzare gli oggetti coinvolti e i messaggi scambiati tra di loro. Le interazioni che verranno trattate, considerate più significative, sono:

1. **Inserimento Squadra**: la sequenza di azioni relativa a quando un utente vuole aggiungere una nuova squadra all'elenco del torneo.
2. **Inserimento Partita**: la sequenza di azioni necessaria per registrare un nuovo match nello storico, includendo la selezione delle squadre e il risultato.
3. **Visualizza Squadra**: la sequenza di azioni relativa a quando l'utente vuole consultare le informazioni dettagliate e la rosa dei giocatori di una singola squadra.
4. **Modifica Squadra**: la sequenza di azioni necessaria per variare i dati di una squadra (nome o lista giocatori), avviata dalla finestra di visualizzazione.
5. **Modifica Partita**: la sequenza di azioni relativa alla rettifica dei gol segnati in un match già disputato, attivabile dalla finestra dello storico.
6. **Ricalcolo Classifica**: una vista dettagliata sulla logica di aggiornamento dinamico dei punteggi, che garantisce la coerenza tra lo storico dei match e la graduatoria del torneo.

4.1) Inserimento Squadra

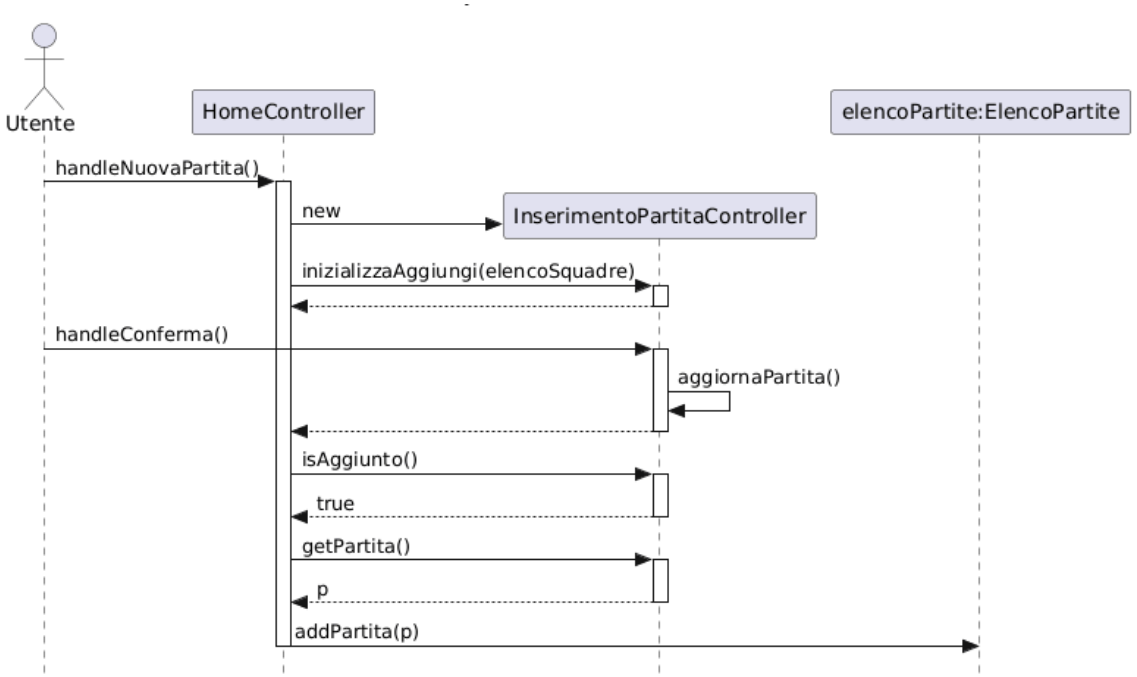


Questo diagramma mostra l'operazione di aggiunta di una nuova squadra al torneo. Sono coinvolti tre attori:

- **Utente**: interagisce con l'interfaccia mediante la pressione dei bottoni.
- **HomeController**: gestisce la finestra principale e coordina l'apertura delle sottofinestre.
- **InserimentoSquadraController**: gestisce la finestra relativa all'aggiunta e la validazione dei dati.
- **elenco: ElencoSquadre**: oggetto che implementa la lista globale delle squadre e gestisce la logica dei dati.

L'operazione inizia con `handleAggiungiSquadra()`. Il controller secondario riceve il riferimento all'elenco per poter effettuare, all'interno del metodo `aggiornaSquadra()`, la verifica dei duplicati tramite `getElencoBase()`. Solo se la verifica ha esito positivo, l'operazione prosegue. L'**HomeController** verifica tramite `isAggiunto()` l'esito della finestra modale e, in caso positivo, recupera l'oggetto tramite `getSquadra()` per aggiungerlo definitivamente all'elenco.

4.2) Inserimento Partita

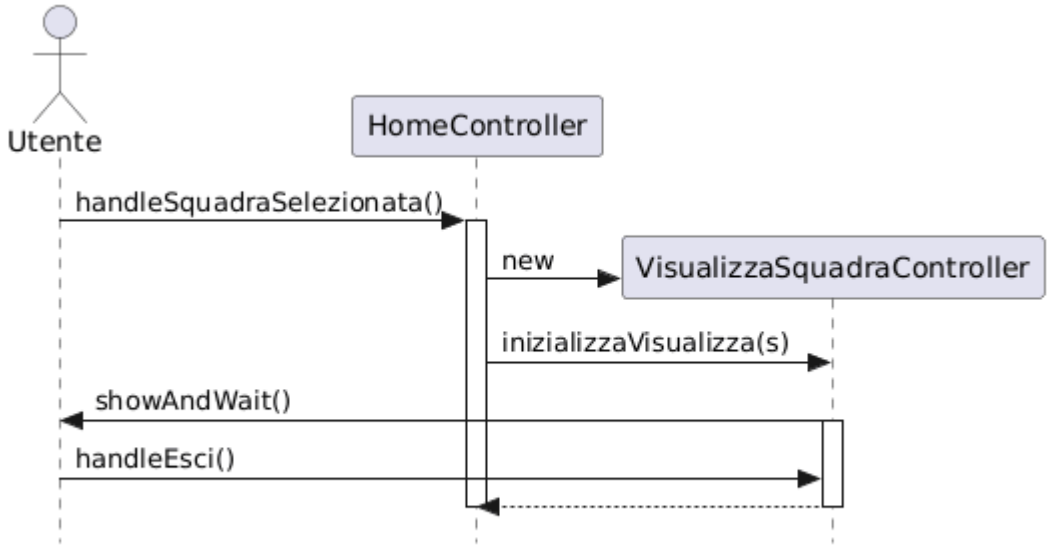


Il diagramma illustra la creazione di un nuovo match nel torneo. Sono presenti quattro attori:

- **Utente**: seleziona le squadre e inserisce il risultato.
- **HomeController**: coordina l'inserimento dalla schermata principale.
- **InserimentoPartitaController**: gestisce il modulo di inserimento della partita.
- **elencoPartite: ElencoPartite**: gestisce lo storico dei match disputati.

L'**HomeController** istanzia l'**InserimentoPartitaController** passandogli l'elenco delle squadre esistenti. Questo passaggio è necessario per permettere all'utente di selezionare le squadre tramite ComboBox. Una volta confermati i dati (`handleConferma`), viene generato l'oggetto **Partita**, che viene successivamente aggiunto all'**elencoPartite** per la persistenza dei dati.

4.3) Visualizza Squadra

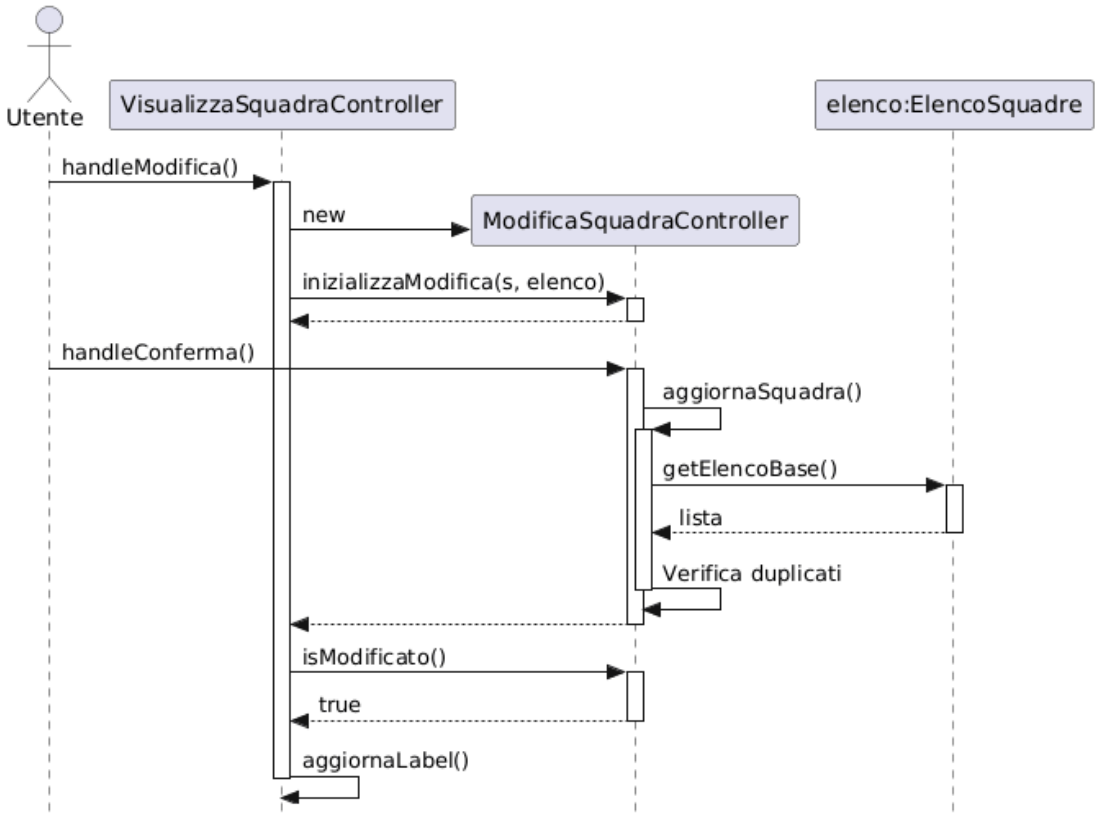


Questo diagramma descrive l'operazione di consultazione dei dettagli di una squadra. Sono presenti tre attori:

- **Utente**: clicca su una riga della tabella dei contatti/squadre.
- **HomeController**: gestisce la finestra principale della rubrica.
- **VisualizzaSquadraController**: gestisce la finestra relativa alla visualizzazione delle informazioni dettagliate.

L'utente attiva il metodo `handleSquadraSelezionata()`. L'**HomeController** crea il **VisualizzaSquadraController** e gli passa l'oggetto selezionato (`s`) tramite `inizializzaVisualizza()`. Il controller secondario popola l'interfaccia e la finestra viene mostrata in modalità `showAndWait()` fino alla chiusura da parte dell'utente.

4.4) Modifica Squadra

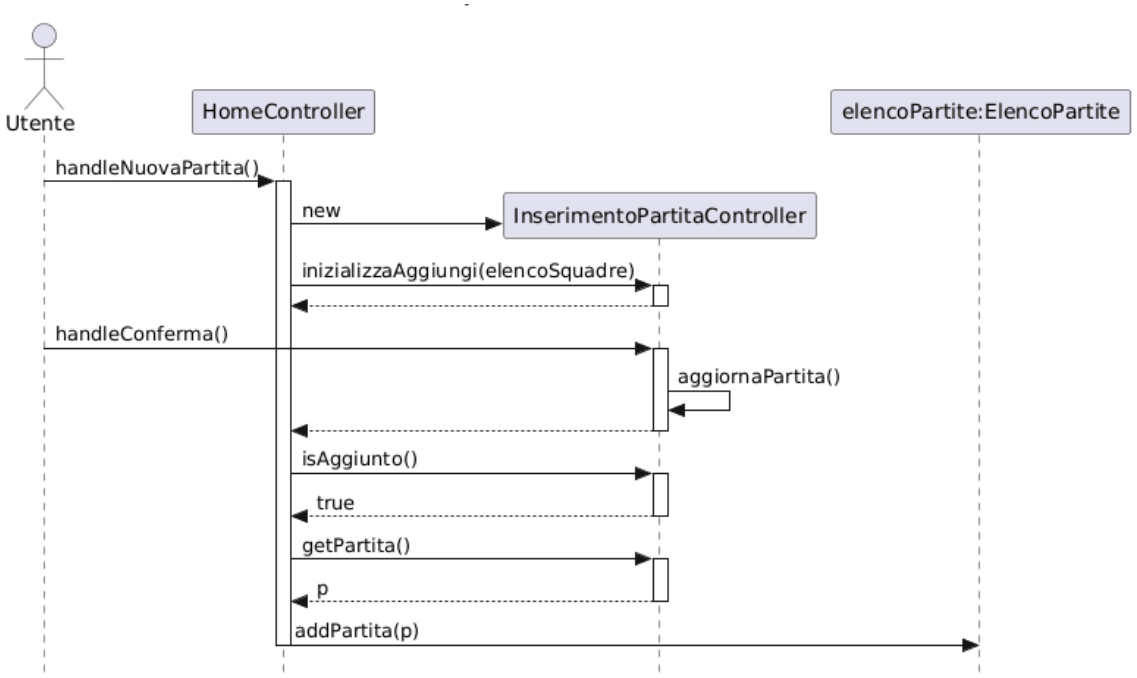


La sequenza descrive la modifica di una squadra esistente, avviata dalla finestra di visualizzazione. Gli attori coinvolti sono:

- **Utente**: effettua le modifiche ai dati nei campi di testo.
- **VisualizzaSquadraController**: gestisce la finestra di dettaglio e riceve l'aggiornamento.
- **ModificaSquadraController**: gestisce la finestra di editing e la validazione.
- **elenco: ElencoSquadre**: utilizzato per verificare che il nuovo nome non sia già occupato da altre squadre.

All'atto della conferma, il controller di modifica esegue `aggiornaSquadra()`, che verifica l'assenza di nomi duplicati consultando l'**ElencoSquadre**. Alla chiusura della finestra, il controller chiamante verifica lo stato tramite `isModificato()` e aggiorna le proprie etichette informative (`aggiornaLabel()`).

4.5) Modifica Partita

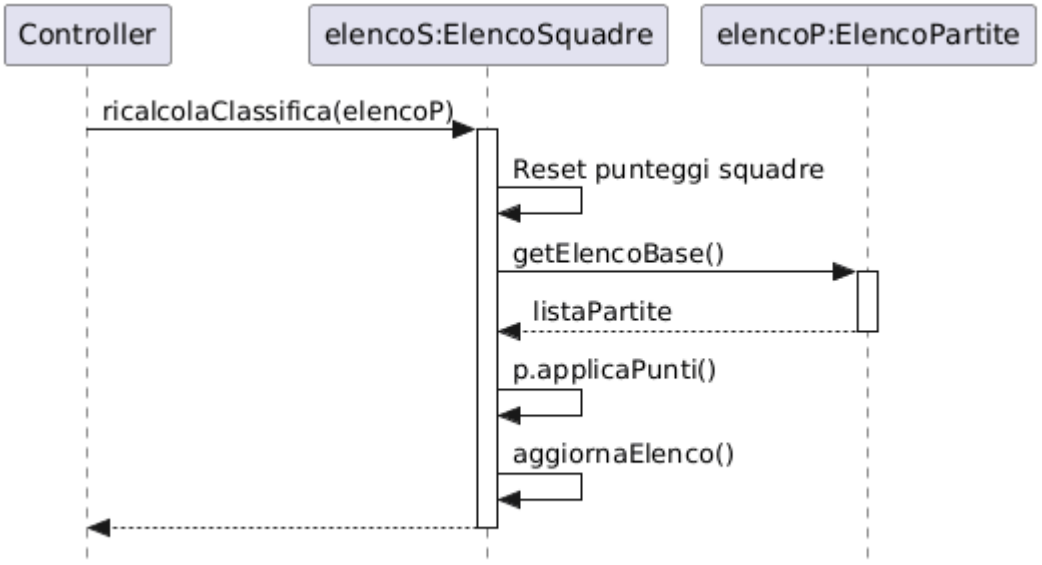


Questo diagramma mostra la modifica dei risultati di una partita nello storico. Coinvolge tre attori:

- **Utente**: modifica i gol segnati.
- **StoricoController**: gestisce la lista di tutte le partite passate.
- **ModificaPartitaController**: gestisce l'interfaccia di modifica del punteggio.

L'operazione ha inizio nello **StoricoController**. Viene creato il **ModificaPartitaController**, a cui viene passata la partita da editare. Dopo la conferma, il metodo `aggiornaPartita()` modifica i dati dell'oggetto originale. Al ritorno del controllo, lo **StoricoController** esegue un `refresh()` della tabella per riflettere immediatamente i cambiamenti.

4.6) Ricalcolo Classifica



Il diagramma descrive la logica di aggiornamento dinamico dei punteggi basata sui risultati delle partite. Sono presenti tre attori logici:

- **Controller**: (Home o Storico) che innesca l'aggiornamento.
- **elencoS: ElencoSquadre**: gestisce la classifica e l'ordinamento.
- **elencoP: ElencoPartite**: fornisce i dati storici dei match.

Il processo prevede tre fasi:

1. **Reset**: l'azzeramento dei punteggi di tutte le squadre per garantire un calcolo pulito.
2. **Iterazione**: il recupero delle partite e l'applicazione dei punti per ogni match tramite `p.applicaPunti()`.
3. **Ordinamento**: l'esecuzione di `aggiornaElenco()` per riordinare la lista in base ai nuovi punteggi, garantendo la coerenza della classifica mostrata all'utente.

5) Scelte progettuali

In fase di progettazione, abbiamo deciso di gestire il torneo attraverso **finestre multiple** invece di un'unica schermata statica, per migliorare l'esperienza utente e la manutenibilità del codice. Inizialmente, il design presentava riferimenti diretti tra controller, ma questa soluzione violava il principio K.I.S.S. (Keep It Simple, Stupid!) e creava un accoppiamento troppo elevato.Abbiamo quindi implementato l'uso di **flag (variabili booleane)** per comunicare lo stato tra le finestre (es. `isModificato`, `isEliminato`), rendendo il codice più leggibile e manutenibile. Per rispettare il principio D.R.Y. (Don't Repeat Yourself), abbiamo notato che sia la Home che lo Storico necessitavano di tabelle: è stata quindi creata la classe base `TabellaController` per gestire tutte le operazioni comuni di visualizzazione. Ogni componente è stato pensato seguendo la **Separation of Concerns**, delimitando i compiti di ricerca, inserimento e ricalcolo punti

5.1) Considerazione sui principi di buona progettazione

Il progetto è stato rifinito seguendo i principi **S.O.L.I.D.**:

- **Singola Responsabilità (SRP)**: Ogni classe si occupa di un unico aspetto, come `IOFile` per la persistenza o `Partita` per il calcolo dei punti.
- **Principio Aperto/Chiuso (OCP)**: Grazie a `TabellaController`, è possibile aggiungere nuove tipologie di visualizzazioni tabellari senza modificare la logica esistente.
- **Sostituzione di Liskov (LSP)**: Le classi derivate (es. `InserimentoSquadraController`) possono sostituire le classi base (`ModificaSquadraController`) senza alterare il comportamento atteso del sistema.
- **Segregazione delle Interfacce (ISP)**: Non sono state utilizzate interfacce sovradimensionate; ogni classe implementa solo i metodi necessari al suo ruolo.
- **Inversione delle Dipendenze (DIP)**: I moduli di alto livello (Controller) interagiscono con la logica tramite astrazioni e parametri, evitando dipendenze dirette da variabili globali e minimizzando l'impatto di eventuali modifiche alla gestione dell'elenco.

.