Iterazione 1

1.0 Introduzione

La prima iterazione si basa sull'analisi e progettazione dei primi tre casi d'uso, ritenuti fondamentali ai fini del funzionamento dell'applicativo.

Scopo di questa, così come delle iterazioni che verranno dopo, è quello di andare ad Implementare in maniera iterativa l'applicativo software raffinando costantemente la sua Visione, identificando e implementando la maggior parte dei requisiti richiesti e test per garantire il corretto funzionamento del codice dell'applicazione.

In particolare, in questa prima iterazione andremo ad implementare gli scenari principali di successo dei primi tre casi d'uso:

- **UC1**: Creazione Torneo
- **UC2**: Tesseramento giocatori
- **UC3**: Iscrizione Torneo

1.1 Aggiornamento dei casi d'uso

Durante questa fase sono stati specificati e rivisti dei dettagli dei casi d'uso implementati. Per visionare le modifiche apportate ai suddetti, si consulti il documento "Modello dei casi d'uso" riportato nella cartella in versione aggiornata per questa iterazione. Se verranno apportate ulteriori modifiche anche a documenti precedenti, queste saranno tramite cronologia.

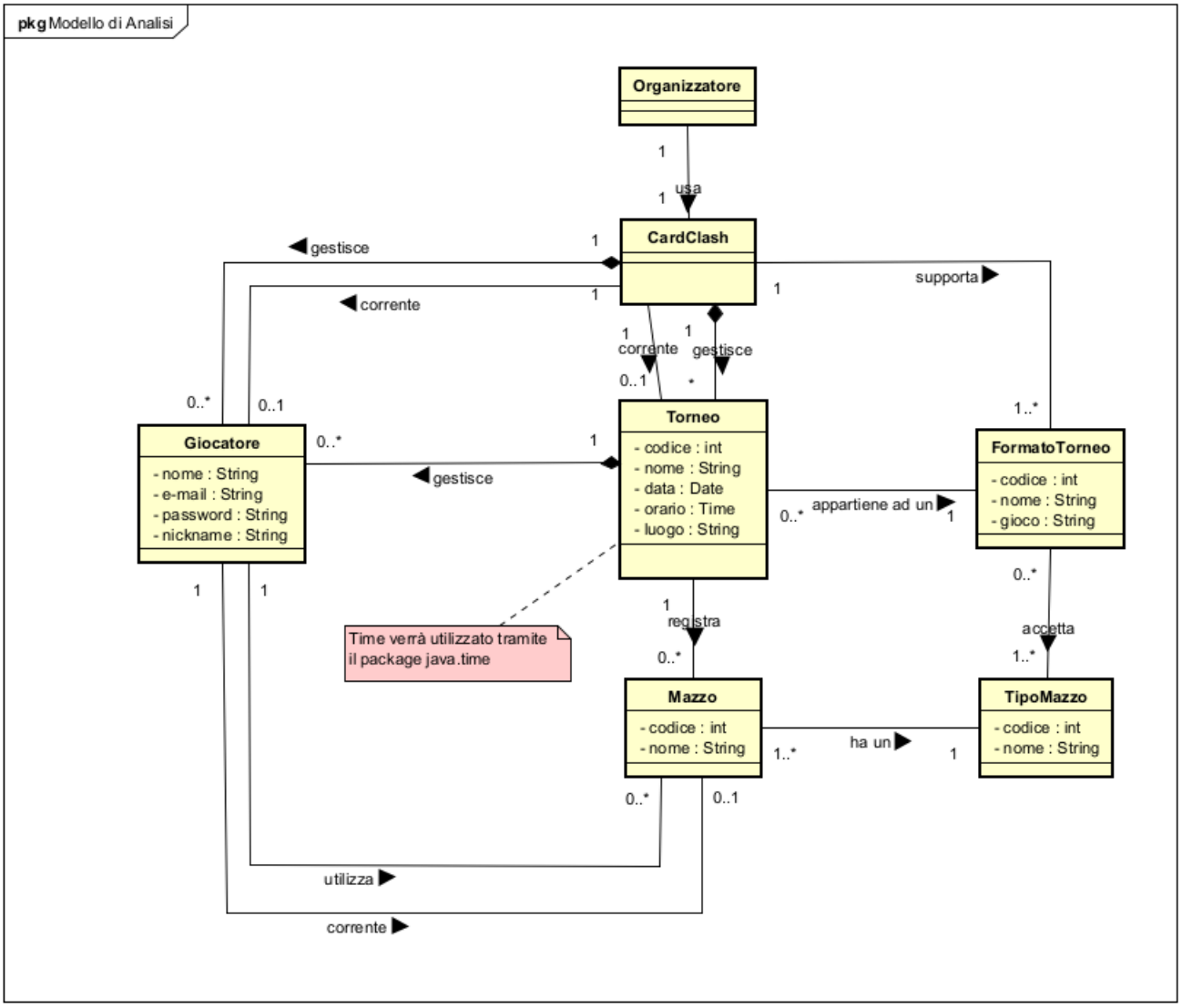
2.0 Fase di Analisi

Le classi concettuali identificate durante questa prima iterazione sono:

- **Organizzatore:** Attore primario che interagisce con il sistema.
- **CardClash:** Rappresenta il sistema CardClash.

- **Torneo:** Rappresenta un torneo, e le relative informazioni come l'id di identificazione, gestite dal sistema.
- **FormatoTorneo:** Tipologie di torneo supportate dal sistema.
- **Giocatore:** Utilizzatore del sistema CardClash, che può effettuare l'iscrizione al torneo e parteciparci.
- **Mazzo:** Rappresenta il Mazzo presentato dall'utente durante l'iscrizione ad un torneo.
- **TipoMazzo:** Rappresenta la tipologia a cui appartiene un Mazzo iscritto e supportato dal sistema.

Da cui tenendo conto di associazioni e attributi, ottenuti analizzando i primi tre casi d'uso, è stato ricavato il seguente Modello di Dominio:

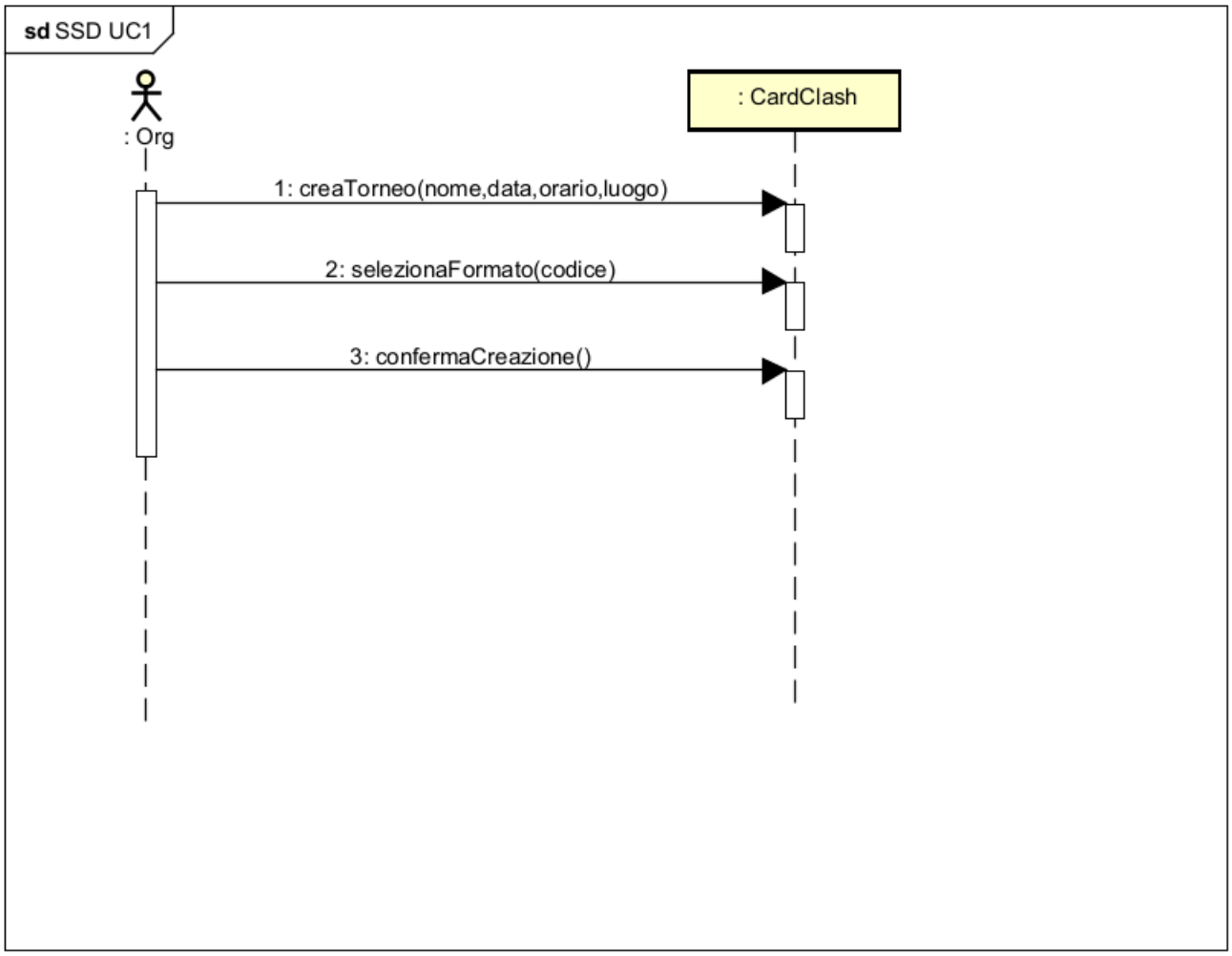


Per come è stato pensato CardClash, non è necessario avere una classe concettuale per i giochi di carte; infatti, essi sono solamente utili per distinguere i formati supportati dal sistema e distinguerli da eventuali loro simili applicati ad un altro gioco (ad esempio, un torneo solo carte comuni su Pokèmon e uno su Yu-Gi-Oh! saranno due formati di torneo diversi). Per tale ragione, abbiamo messo un codice identificativo per facilitare la ricerca del formato nel sistema.

2.1 Diagrammi di sequenza di sistema

- **UC1:** Creazione Torneo

Il seguente Diagramma di Sequenza di Sistema (SSD) illustra il corso di eventi I/O tra l'organizzatore e il sistema CardClash.



2.1.1 Contratti delle operazioni

I seguenti Contratti descrivono le principali operazioni eseguite dal sistema sulla base degli eventi individuati nell'SSD.

Contratto1 UC1: Creazione Torneo

- **Operazione:** creaTorneo(nome,data,orario,luogo)
- **Riferimenti:** Caso d'uso: Creazione Torneo
- **Pre-condizione:** L'organizzatore è autenticato e ha accesso ai permessi di creazione.
- **Post-condizione:**
  - È stata creata un'istanza t di Torneo
  - Gli attributi di t vengono inizializzati
  - t diventa corrente per CardClash

Contratto2 UC1: Creazione Torneo

- **Operazione:** selezionaFormato(codice)
- **Riferimenti:** Caso d'uso: Creazione Torneo
- **Pre-condizione:**
  - È stata già creata una nuova istanza t di torneo
  - Gli attributi di t sono stati inizializzati
  - t è corrente per CardClash
- **Post-condizione:**
  - Viene inserito il formato f per il torneo t corrente

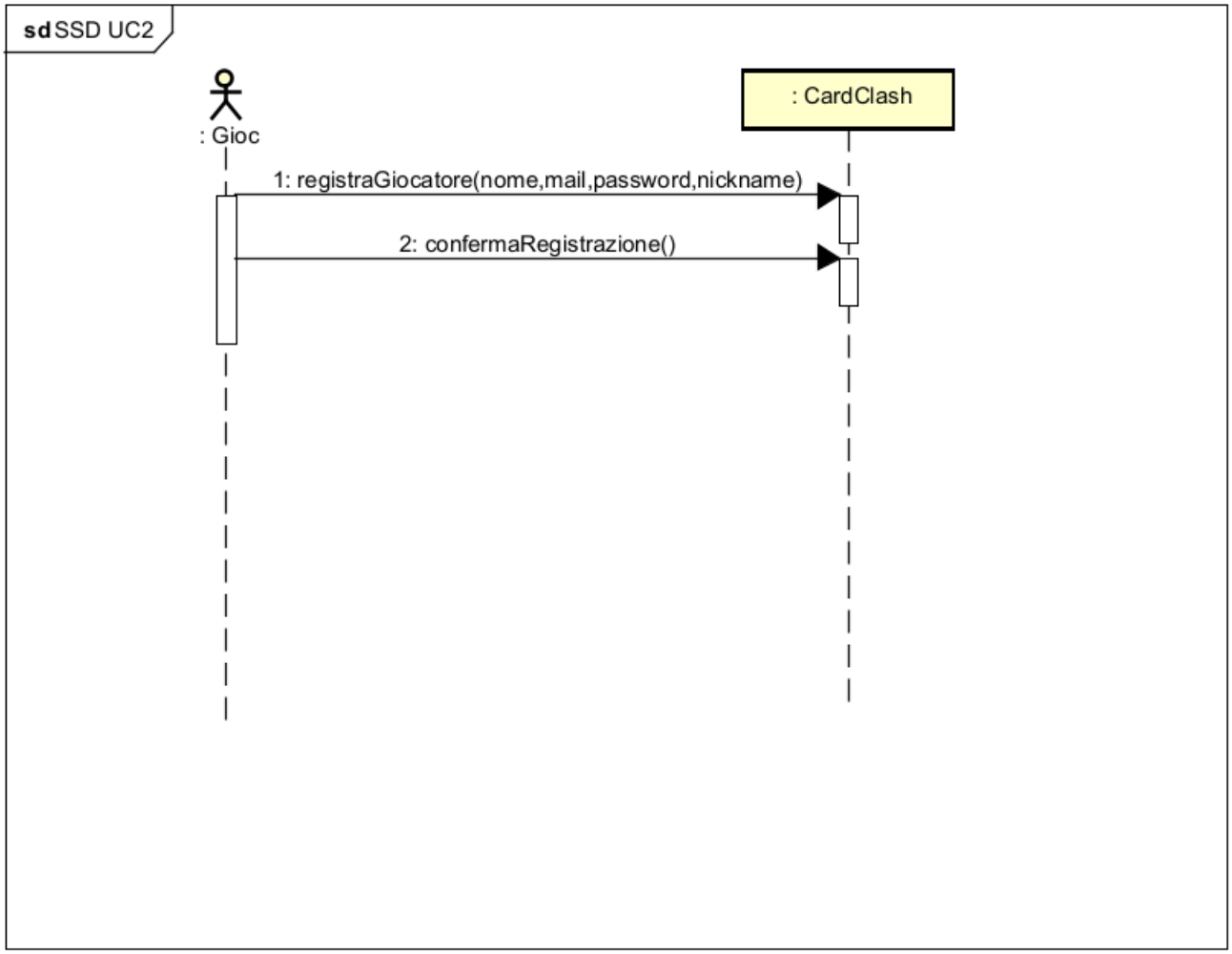
Contratto3 UC1: Creazione Torneo

- **Operazione:** confermaCreazione()
- **Riferimenti:** Caso d'uso: Creazione Torneo
- **Pre-condizione:** Gli attributi di t sono stati inizializzati correttamente e gli è stato associato il formato f di torneo selezionato
- **Post-condizione:**
  - Viene associato a t un id univoco
  - Si salva l'istanza t di Torneo, e la si associa a CardClash

2.2 Diagrammi di sequenza di sistema

- **UC2:** Tesseramento Giocatori

Il seguente Diagramma di Sequenza di Sistema (SSD) illustra il corso di eventi I/O tra l'organizzatore e il sistema CardClash.



2.2.1 Contratti delle operazioni

I seguenti Contratti descrivono le principali operazioni eseguite dal sistema sulla base degli eventi individuati nell'SSD.

Contratto1 UC2: Tesseramento Giocatori

- **Operazione:** registraGiocatore(nome,mail,password,nickname)
- **Riferimenti:** Caso d'uso: Tesseramento Giocatori
- **Pre-condizione:**
  - Il giocatore non deve essere già registrato
- **Post-condizione:**
  - È stata creata un'istanza g di giocatore
  - Sono stati inizializzati gli attributi di g
  - g diventa corrente per CardClash

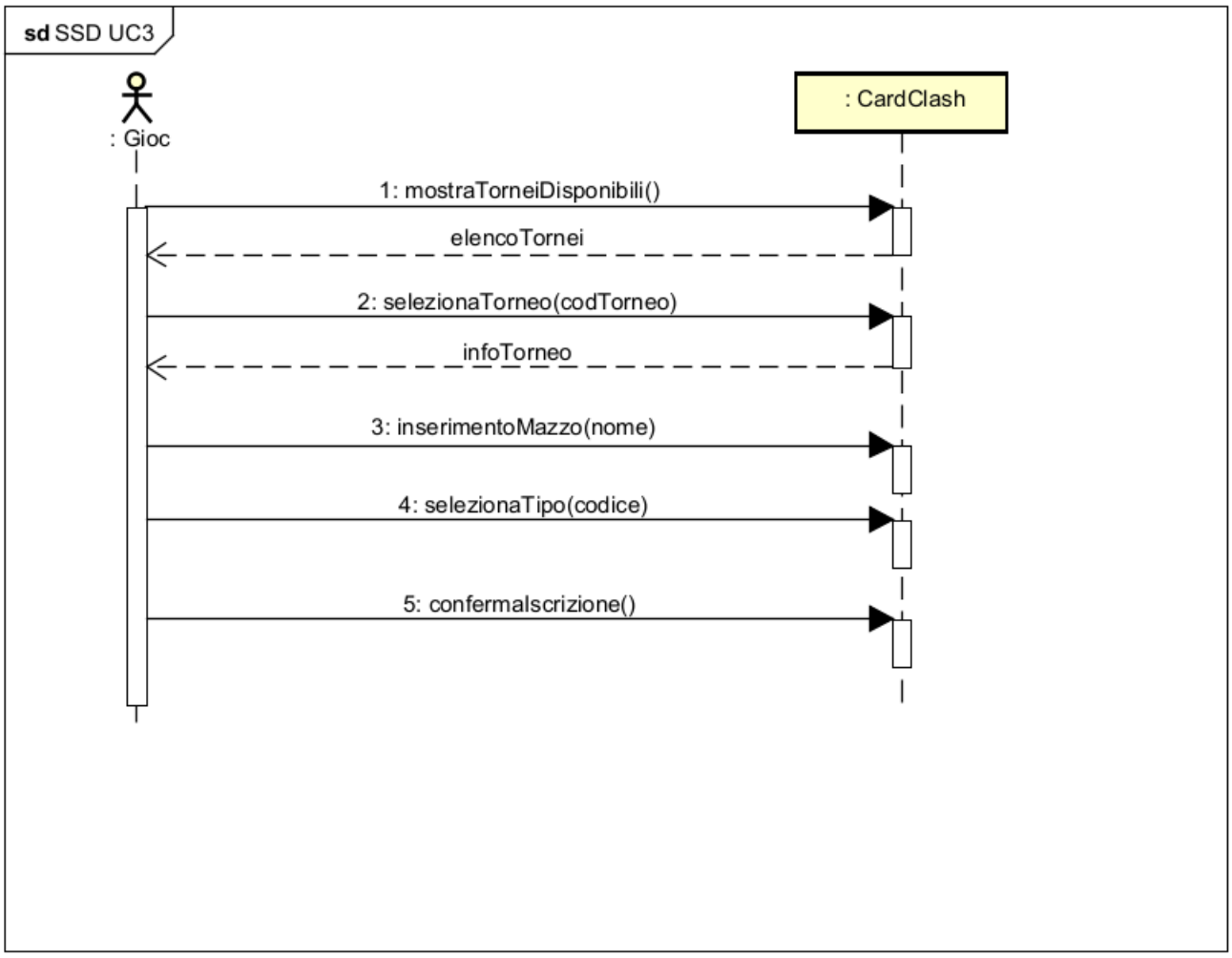
Contratto2 UC2: Tesseramento Giocatori

- **Operazione:** confermaRegistrazione()
- **Riferimenti:** Caso d'uso: Tesseramento Giocatori
- **Pre-condizione:**
  - È stata creata un'istanza g di giocatore
  - g è corrente
- **Post-condizione:**
  - Si salva l'istanza g di giocatore, e la si associa a CardClash

2.3 Diagramma di sequenza di sistema

- **UC3:** Iscrizione Torneo

Il seguente Diagramma di Sequenza di Sistema (SSD) illustra il corso di eventi I/O tra l'organizzatore e il sistema CardClash.



2.3.1 Contratti delle operazioni

Contratto1 UC3: Iscrizione Torneo

- **Operazione:** mostraTorneiDisponibili()
- **Riferimenti:** Caso d'uso: Iscrizione Torneo
- **Pre-condizione:**
  - Sono presenti istanze di torneo
- **Post-condizione:**
  - Viene recuperata la lista dei tornei disponibili a cui è possibile iscriversi

Contratto2 UC3: Iscrizione Torneo

- **Operazione:** selezionaTorneo(codTorneo)
- **Riferimenti:** Caso d'uso: Iscrizione Torneo
- **Pre-condizione:**
  - È in corso un'iscrizione ad un torneo
  - Il giocatore deve essere registrato e autenticato all'interno del sistema

  Post-condizione:
  - Vengono restituite le informazioni del torneo t selezionato

Contratto3 UC3: Iscrizione Torneo

- **Operazione:** inserimentoMazzo(nome)
- **Riferimenti:** Caso d'uso: Iscrizione Torneo
- **Pre-condizione:**
  - Il torneo t selezionato deve essere attivo e aperto ad iscrizioni
- **Post-condizione:**
  - Viene creata un'istanza m di mazzo
  - Gli attributi di m vengono inizializzati
  - m diventa corrente per Giocatore

Contratto4 UC3: Iscrizione Torneo

- **Operazione:** selezionaTipo(codice)
- **Riferimenti:** Caso d'uso: Iscrizione Torneo
- **Pre-condizione:**
  - È stata inizializzata l'istanza m di mazzo e i suoi attributi sono stati settati correttamente
  - Sono state recuperate le tipologie di mazzo consentite per il formato del torneo t selezionato, tramite le quali l'utente potrà associarle ad m
- **Post-condizione:**

- Viene recuperata la tipologia tm di mazzo inserita
- tm viene associata al mazzo m

Contratto5 UC3: Iscrizione Torneo

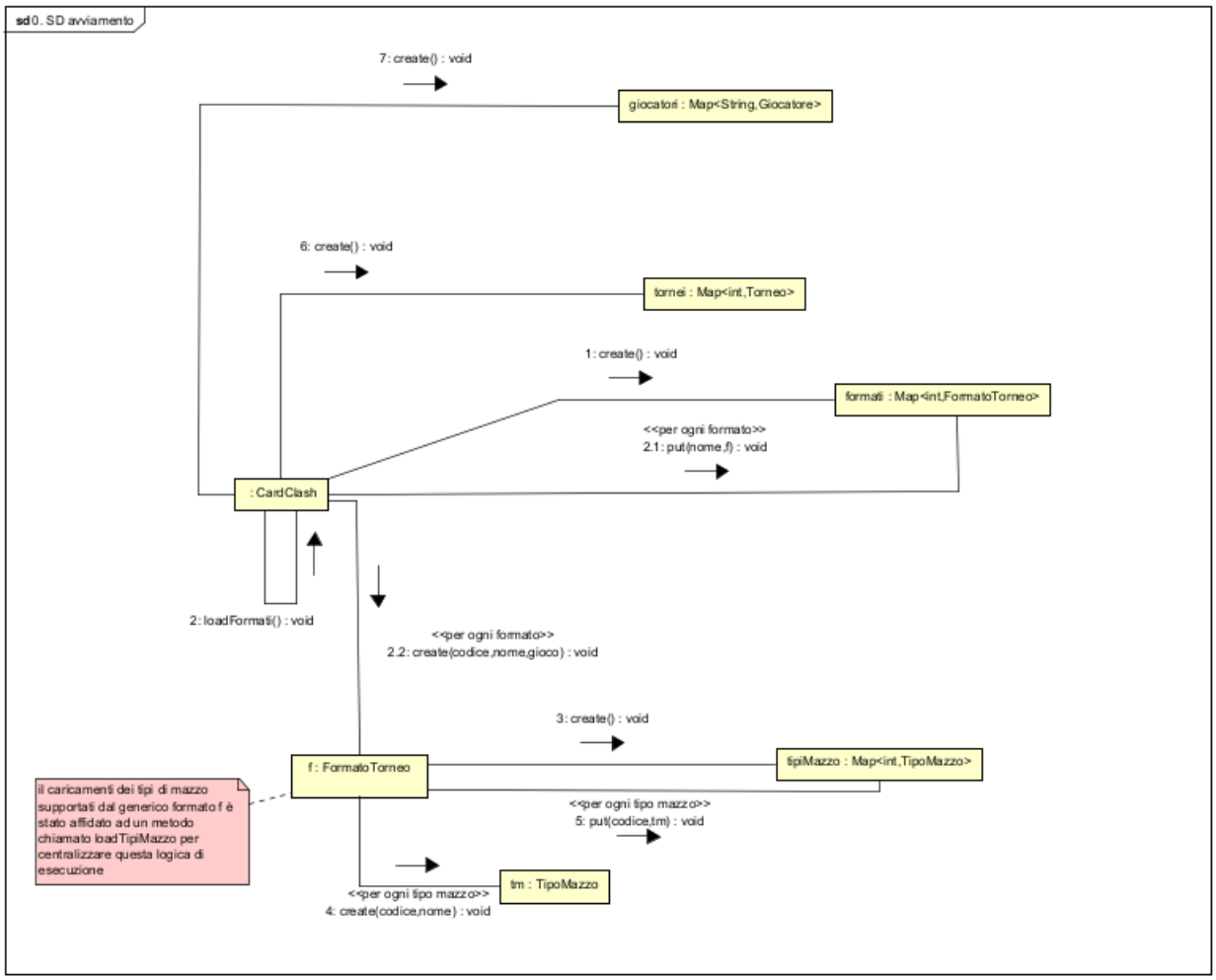
- **Operazione:** confermaIscrizione()
- **Riferimenti:** Caso d'uso: Iscrizione Torneo
- **Pre-condizione:**
  - Gli attributi di m sono stati inizializzati
  - m è corrente per giocatore
- **Post-condizione:**
  - Si salva l'istanza di m e la si associa al giocatore
  - m viene aggiunto alla lista dei mazzi registrati per il torneo t
  - Il giocatore g viene aggiunto alla lista dei partecipanti del torneo t

3.0 Fase di Progettazione

Gli elaborati principali della fase di progettazione sono i diagrammi di interazione, articolati in diagrammi di sequenza e diagrammi di interazione, con lo scopo di descrivere il comportamento del sistema da un punto di vista dinamico durante i casi d'uso presi in considerazione per questa prima iterazione. Insieme a questi, il diagramma delle classi rappresenta il sistema da un punto di vista statico. Di seguito vengono riportati:

3.1 Diagramma di comunicazione (caso d'uso di avviamento)

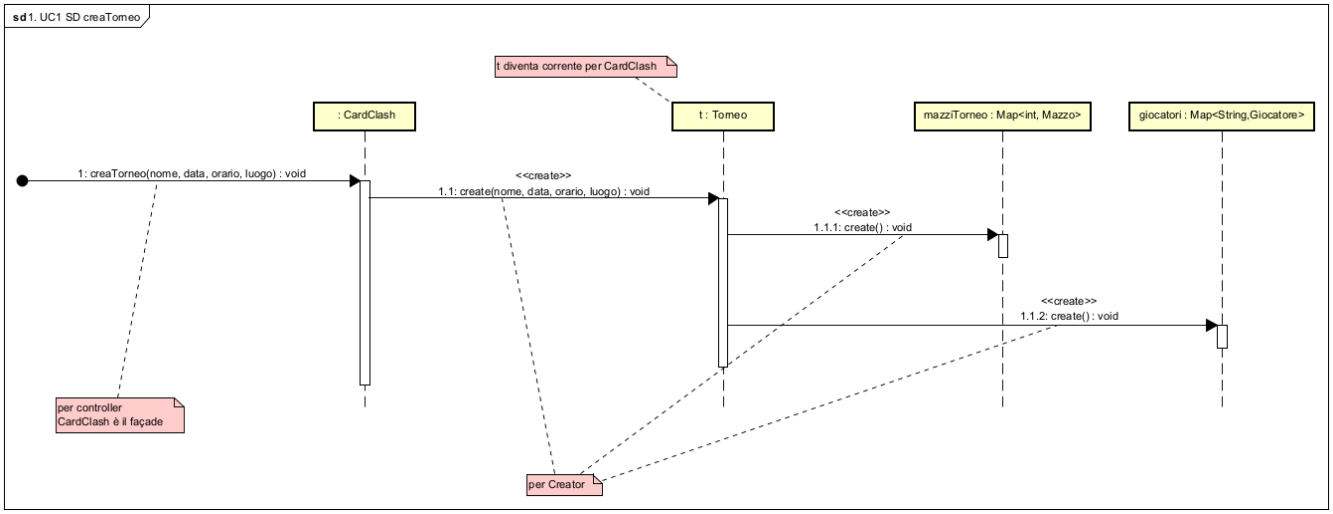
Per funzionare correttamente, il sistema dovrà seguire dei passaggi iniziali di caricamento dati e configurazione relativamente ai formati di torneo supportati da CardClash.



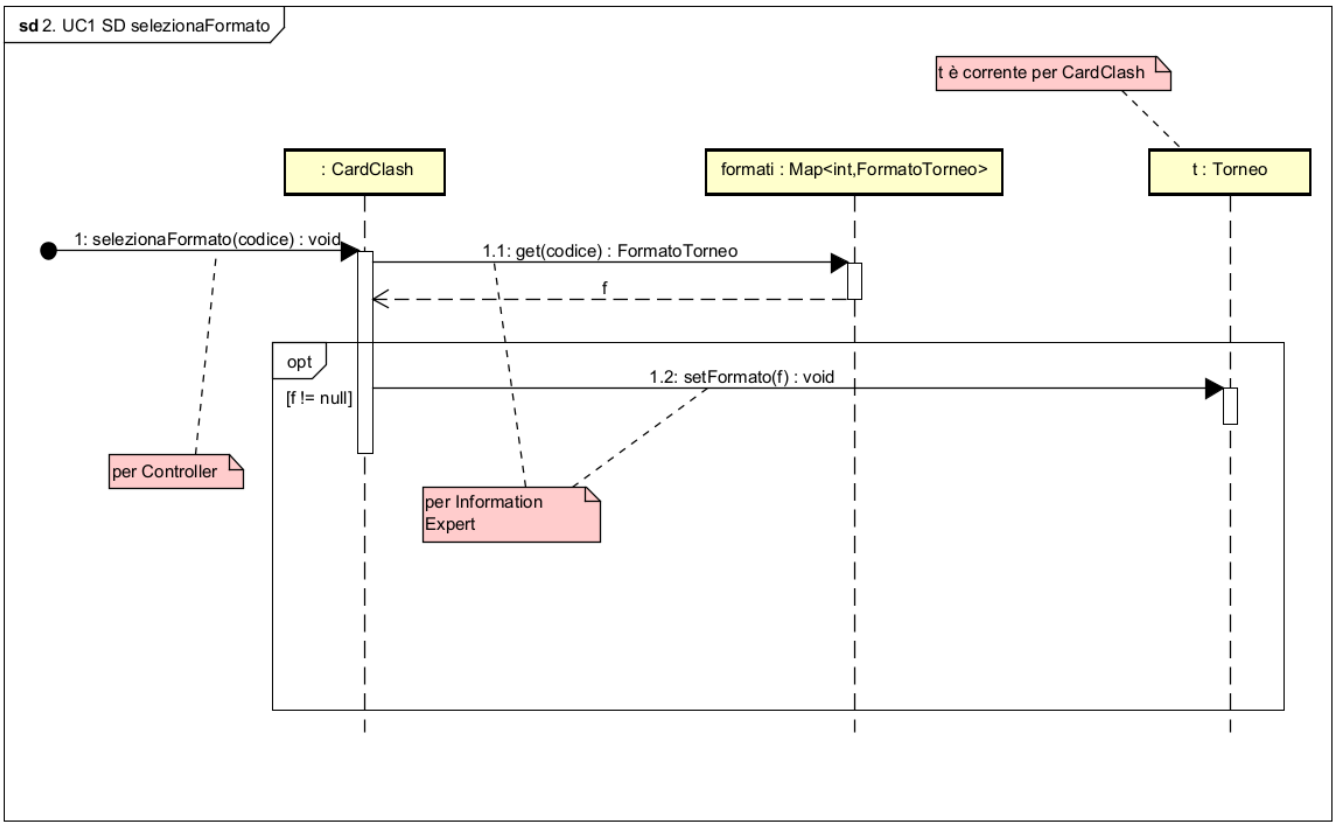
3.2 Diagrammi di sequenza UC1

CardClash è stato progettato secondo il pattern GoF Façade Controller per lavorare come interfaccia del sistema, sia quando viene utilizzato dall'Organizzatore che dai giocatori. Sarà dunque suo il compito di creare (secondo il pattern Creator) le istanze dei tornei.

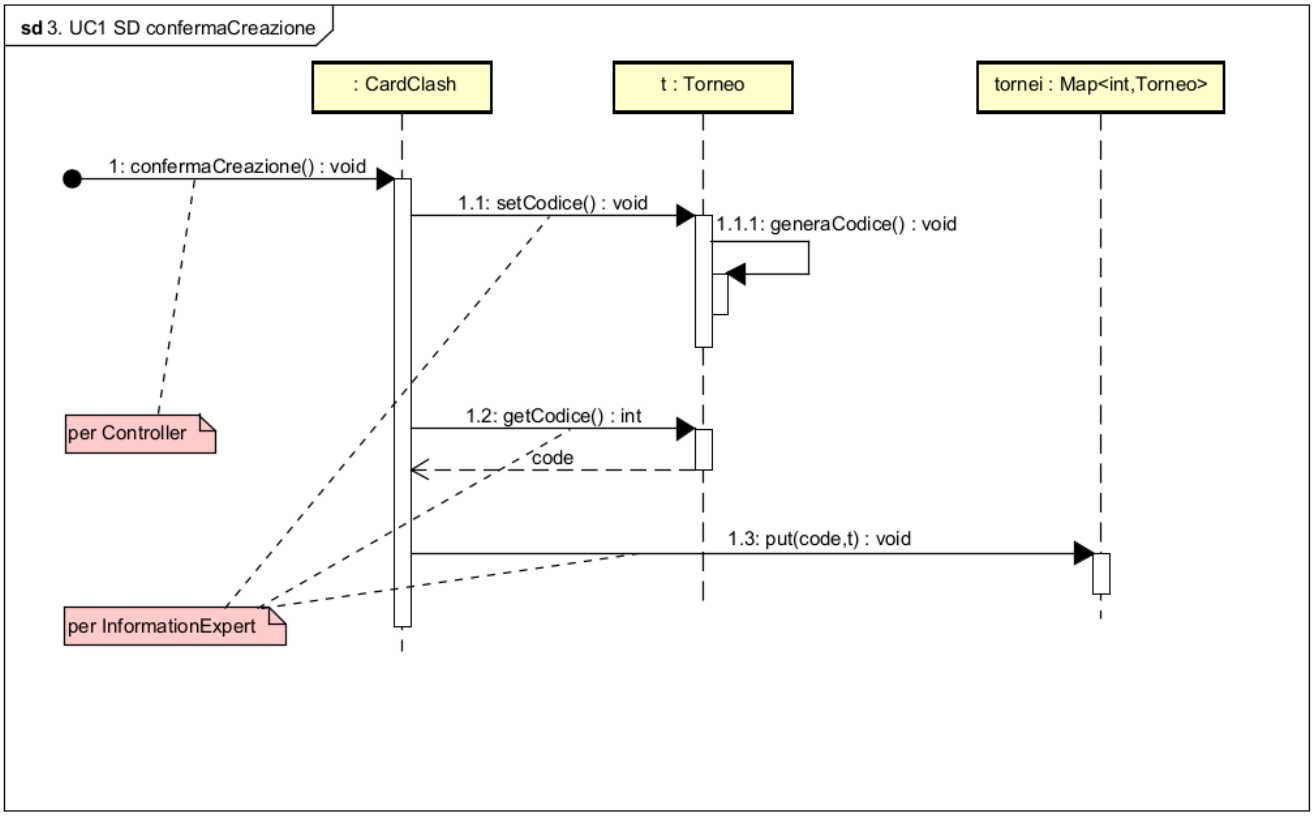
creaTorneo



selezionaFormato



confermaCreazione

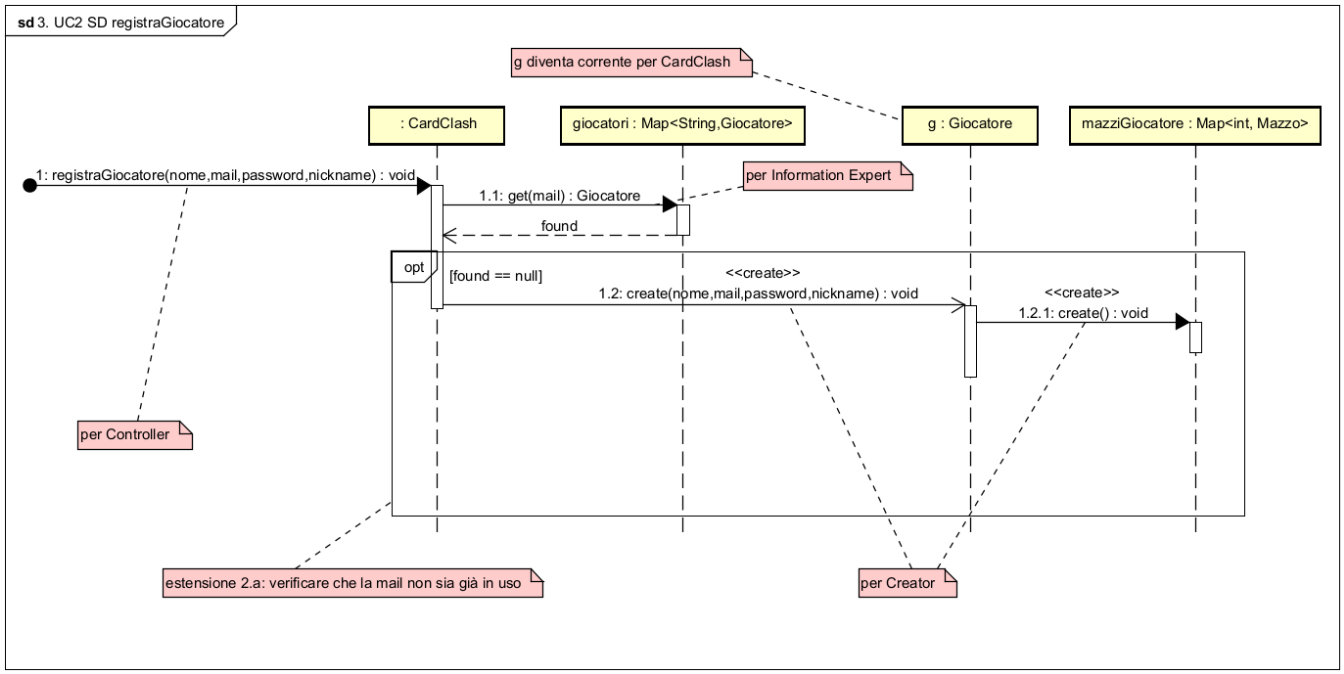


3.3 Diagrammi di sequenza UC2

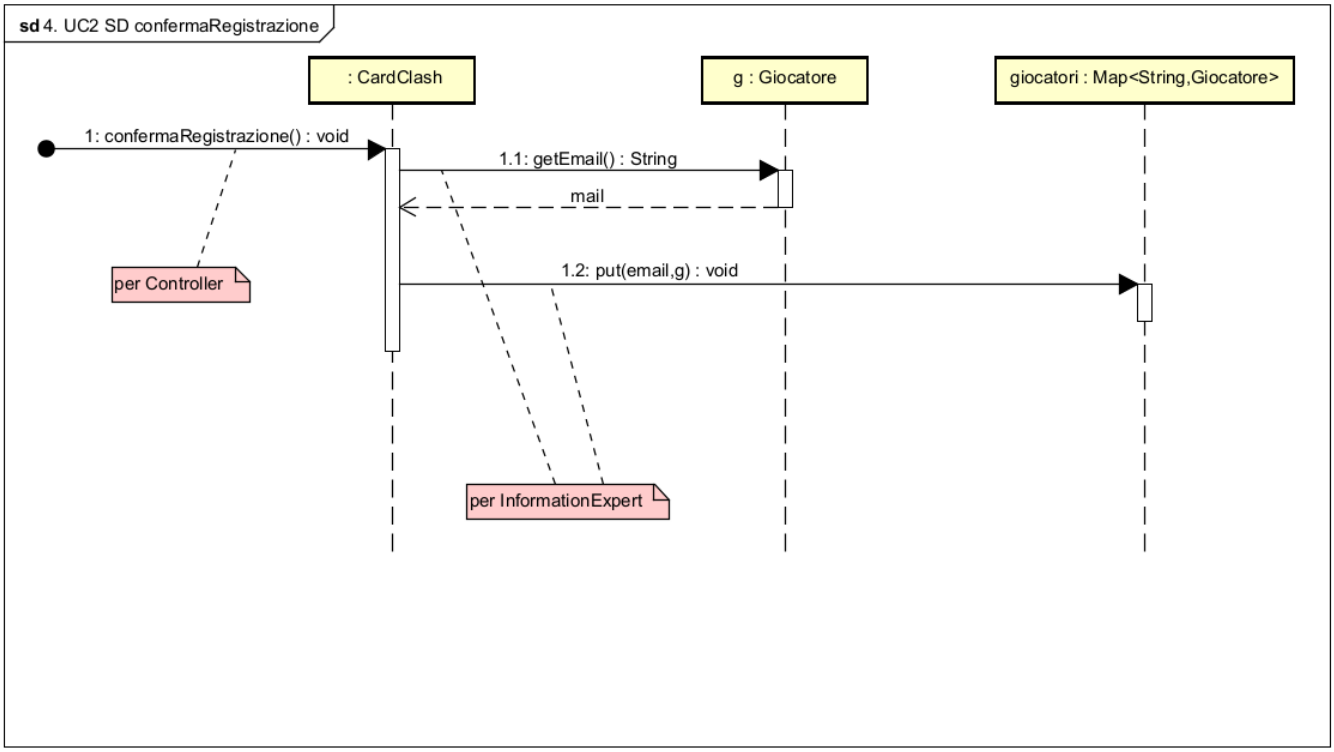
CardClash tiene traccia di tutti i giocatori iscritti all'applicazione, dunque ha la responsabilità di creare nuovi oggetti di tipo Giocatore (per il pattern Creator) e di aggiungerli alla mappa dei Giocatori, che utilizza per tenerne traccia.

Data la semplicità realizzativa, si è deciso di utilizzare sin da subito un'eccezione ad hoc per implementare l'estensione 2.a. Si veda il codice di questa iterazione per approfondire.

registraGiocatore



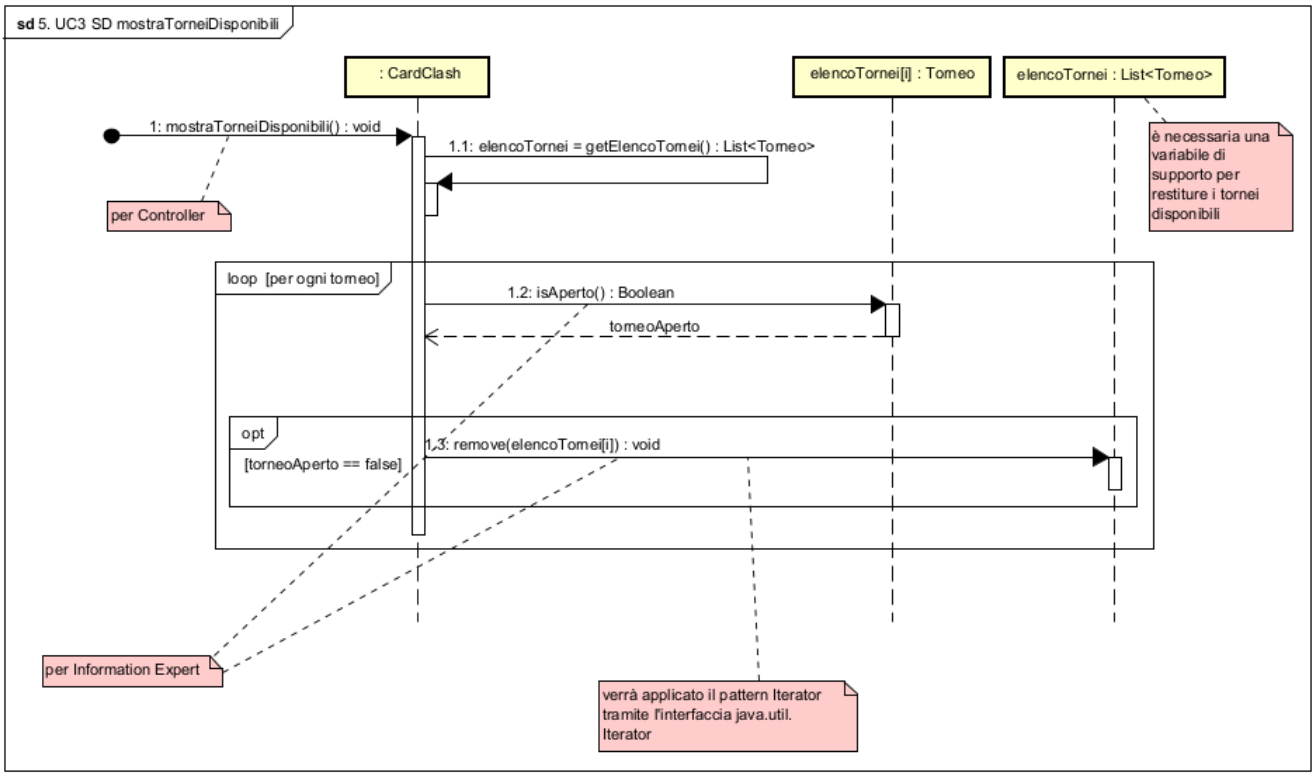
confermaRegistrazione



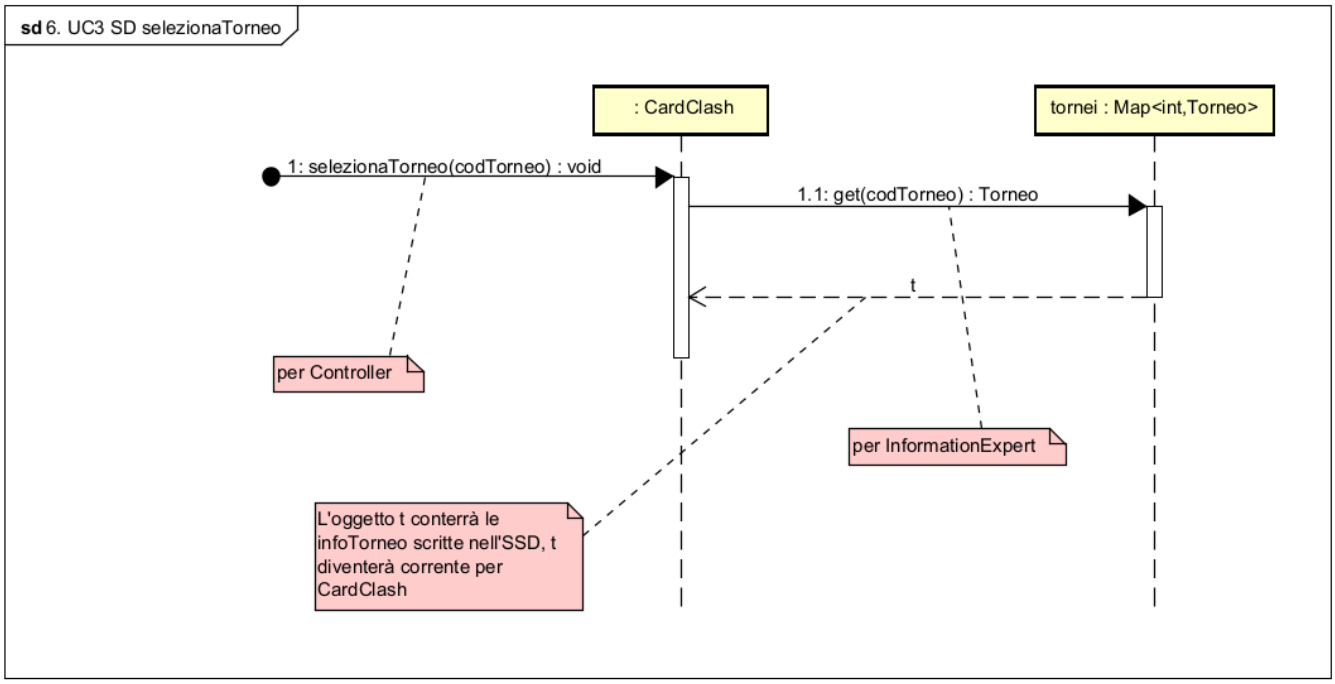
3.4 Diagrammi di sequenza UC3

Questo caso d'uso implementa la possibilità di visualizzare i tornei disponibili, registrare un mazzo con la tipologia a cui appartiene ed iscriversi al torneo.

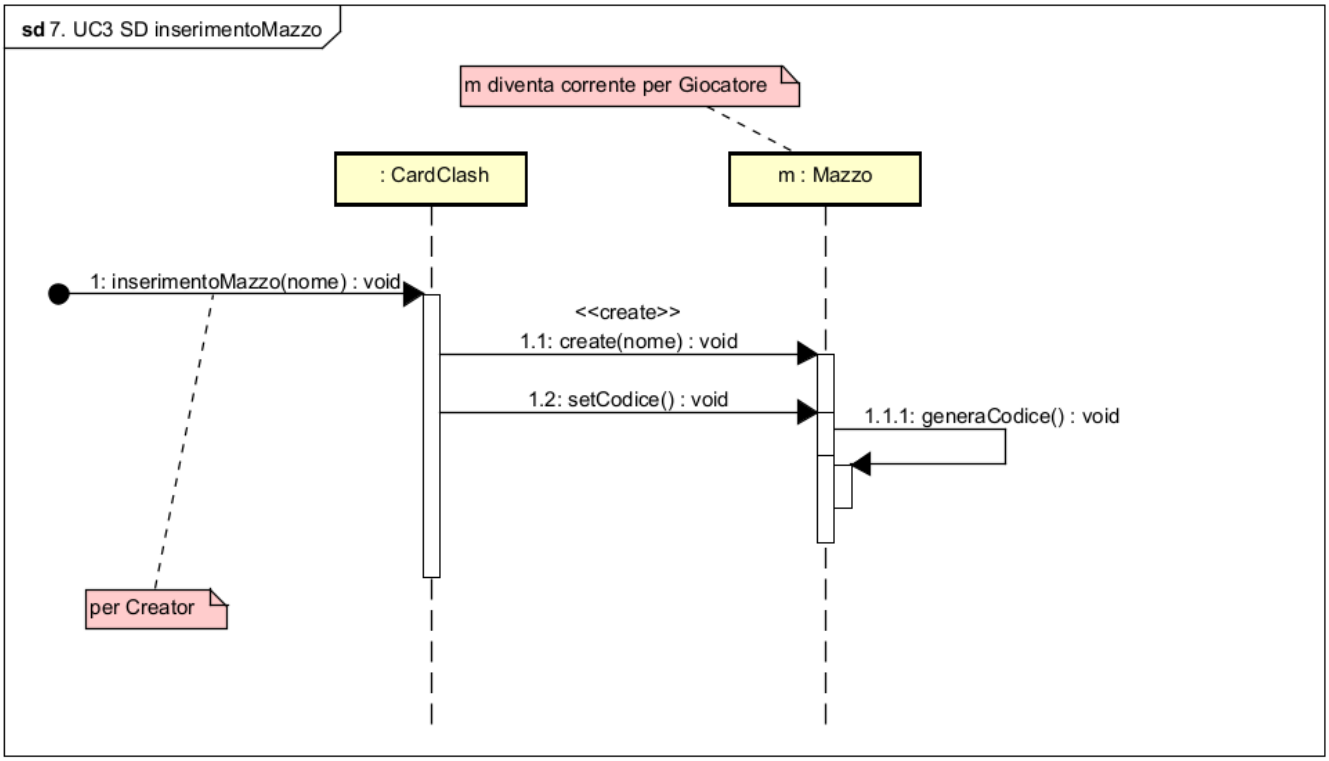
mostraTorneiDisponibili



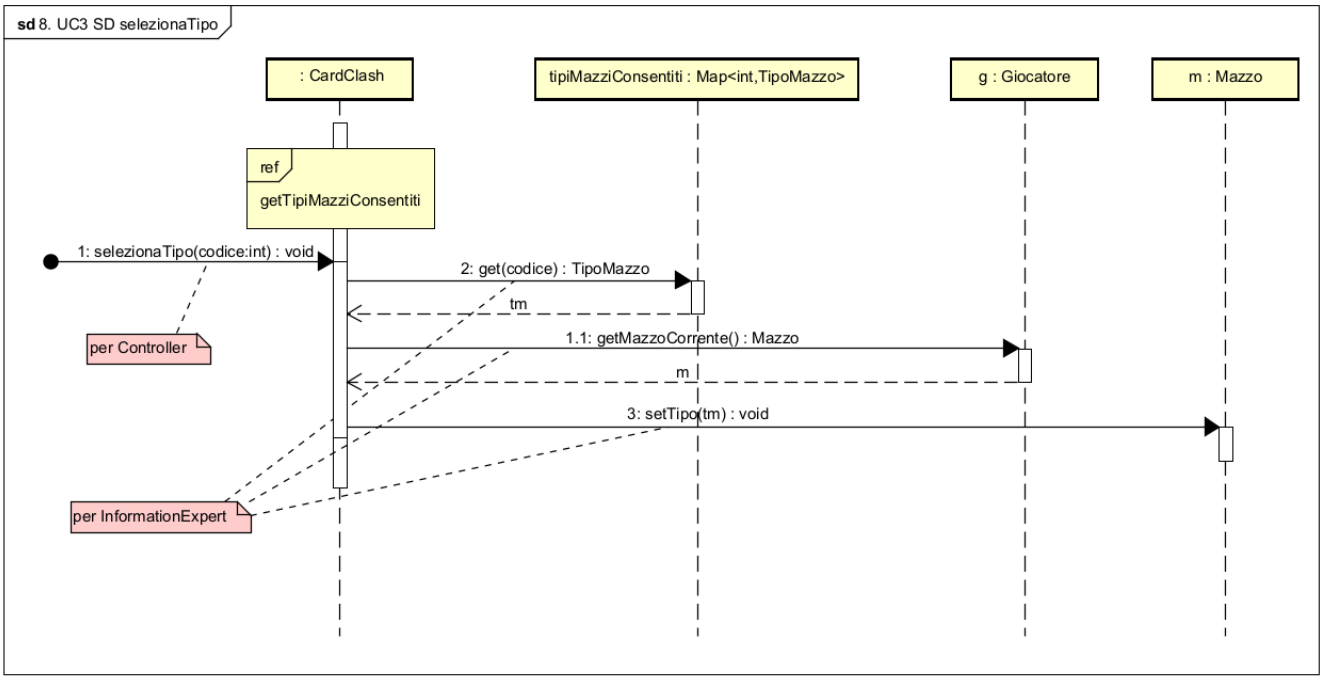
selezionaTorneo



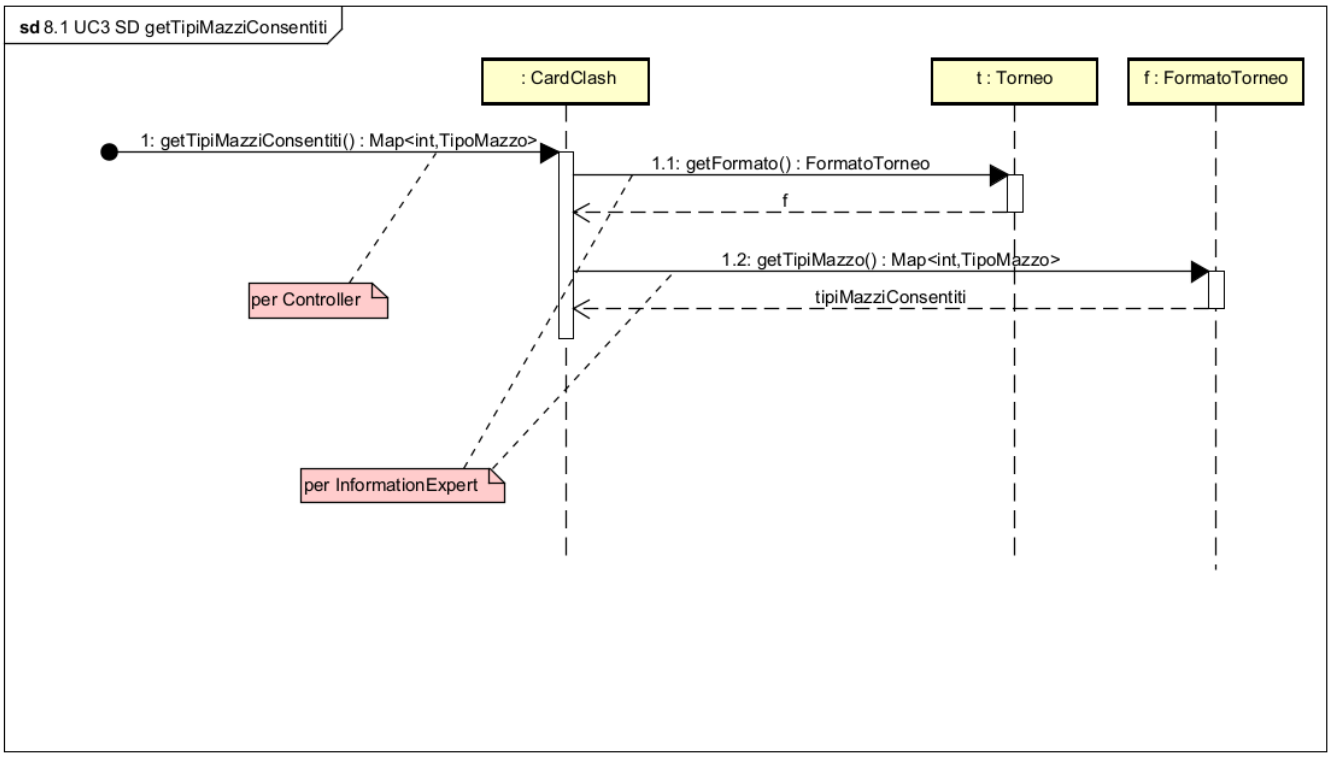
inserimentoMazzo



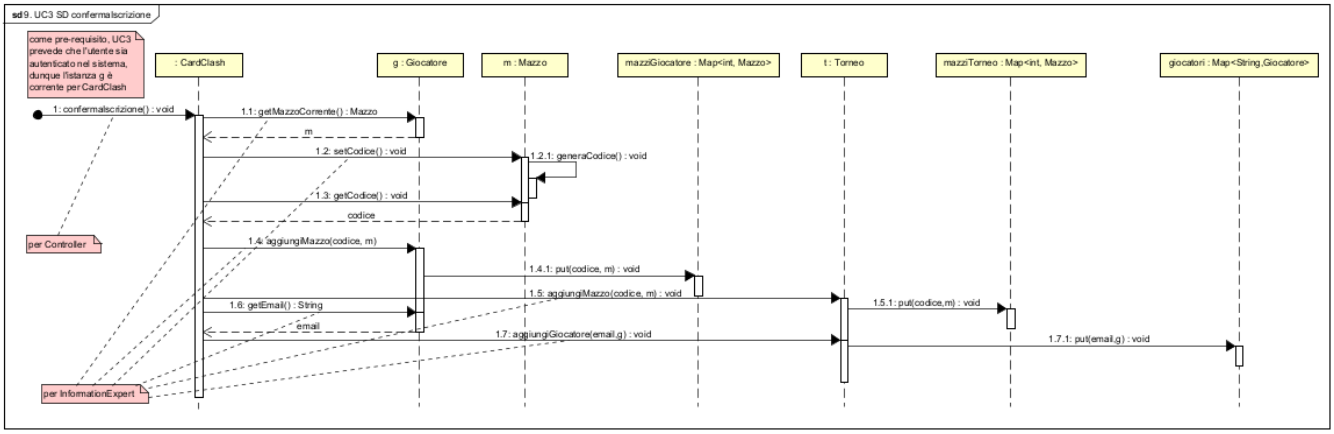
selezionaTipo



getTipiMazzoConsentiti

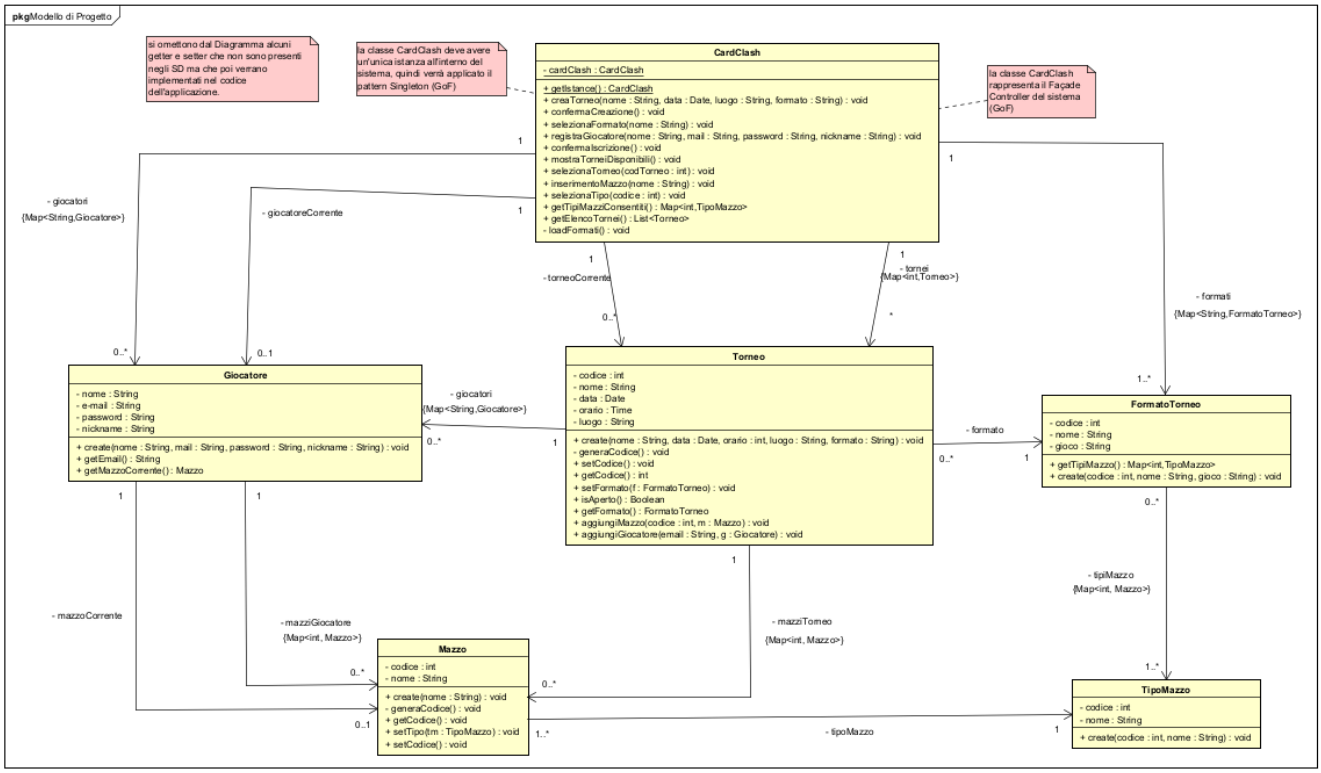


confermaIscrizione



4.0 Diagramma delle classi

Per la classe CardClash si utilizzerà il pattern GoF Singleton e, come detto in precedenza, servirà da Façade Controller per il sistema.



5.0 Testing

5.1 Introduzione

La fase di test è fondamentale nel processo di sviluppo di un programma efficace e robusto; permette la ricerca di errori e anomalie nel software, il che riduce significativamente i costi di manutenzione e migliora la qualità generale del prodotto. Sebbene rendere il progetto di testing tale da coprire ogni caso che si possa presentare sia abbastanza complesso, abbiamo cercato di affinare le tecniche note per ottenere un testing che possa rilevare eventuali comportamenti indesiderati e garantire che il software soddisfi le esigenze del committente.

I test si dividono principalmente in due categorie:

- **Test unitari (Unit Test)**: Sono generalmente associati al white-box testing, perché implicano una conoscenza approfondita del codice sorgente. Gli sviluppatori verificano il funzionamento interno di metodi e classi,

analizzando il flusso di controllo e le strutture dati. Nel nostro progetto, almeno fino a questa iterazione, è stata principalmente utilizzata questa tecnica cercando di seguire un approccio di tipo bottom-up. Gli Unit Test ideati sono stati integrati nel progetto tramite il framework JUnit, che ben si presta a questo scopo. A seguito dell'esecuzione dei test, si è passati alla correzione delle porzioni di codice interessate dagli eventuali fallimenti.

- **Test funzionali**: verificano il funzionamento del sistema software nella sua interezza, trattandolo come una black box, fornendo input e analizzando gli output generati. Questa tipologia di test è preferibile utilizzarla a prodotto ultimato, quindi la valuteremo nel corso delle iterazioni successive.

5.2 Individuazione dei casi di test

In una fase preliminare al testing, si è svolta un'analisi del codice per individuare le classi e i metodi da testare. La strategia è stato quello di dare priorità ai metodi cruciali per il funzionamento del sistema relativamente ai casi d'uso implementati.

Di seguito sono elencati i principali casi di test individuati per le varie classi:

CardClashTest

1. **testSingletonInstance()**
   - Verifica che l'istanza di `CardClash` sia un singleton, ossia che venga restituita la stessa istanza ogni volta che viene richiesta.

2. **testLoadFormati()**
   - Verifica che i formati di torneo siano correttamente caricati e che siano presenti 3 formati.

3. **testCreaTorneo()**
   - Verifica che la creazione di un torneo funzioni e che venga impostato correttamente come torneo corrente.

4. **testSelezionaFormato()**
    - Verifica che il formato del torneo venga selezionato correttamente. Se un formato non esiste, non viene selezionato.

5. **testConfermaCreazione()**
    - Verifica che la conferma della creazione del torneo aggiunga il torneo all'elenco dei tornei.

6. **testRegistraGiocatore()**
    - Verifica che la registrazione di un giocatore avvenga correttamente e che venga sollevata un'eccezione se il giocatore è già registrato.

7. **testConfermaRegistrazione()**
    - Verifica che la conferma della registrazione di un giocatore funzioni e che il giocatore venga aggiunto alla lista dei giocatori.

8. **testMostraTorneiDisponibili()**
    - Verifica che solo i tornei futuri siano visibili tra i tornei disponibili e che non sia possibile iscriversi a tornei con data passata o odierna.
9. **testSelezionaTorneo()**
    - Verifica che un torneo venga selezionato correttamente in base al codice.

10. **testInserimentoMazzo()**
    - Verifica che un mazzo venga inserito correttamente per un giocatore e che venga assegnato un codice identificativo al mazzo.

11. **testSelezionaTipo()**
    - Verifica che il tipo di mazzo venga selezionato correttamente e associato al mazzo. Se il tipo non è valido, il mazzo non avrà tipo.

12. **testConfermaIscrizione()**

- Verifica che la conferma dell'iscrizione di un giocatore a un torneo funzioni, associando correttamente il mazzo del giocatore al torneo

GiocatoreTest

1. **testAggiungiMazzo()**
   - Verifica che la conferma dell'iscrizione di un giocatore a un torneo funzioni, associando correttamente il mazzo del giocatore al torneo
2. **testSetMazzoCorrente()**
   - Verifica che il mazzo corrente di un giocatore venga impostato correttamente. Dopo aver assegnato un mazzo come mazzo corrente, si controlla che il mazzo corrente del giocatore sia effettivamente quello appena impostato.

TorneoTest

1. **testAggiungiGiocatore()**
   - Verifica che un giocatore venga aggiunto correttamente al torneo. Prima di aggiungere il giocatore, si controlla che il numero di giocatori nel torneo sia 0. Dopo aver aggiunto il giocatore, si verifica che la dimensione della mappa dei giocatori sia 1, che il giocatore sia presente e che la mappa dei giocatori non sia null.

2. **testAggiungiMazzo()**
   - Verifica che un mazzo venga aggiunto correttamente al torneo. Prima di aggiungere il mazzo, si controlla che la dimensione della mappa dei mazzi sia 0. Dopo aver aggiunto il mazzo, si verifica che la dimensione della mappa dei mazzi sia 1, che il mazzo sia presente e che la mappa dei mazzi non sia null.

3. **testIsAperto()**

- Verifica se il torneo è aperto in base alla data. Se la data del torneo è futura, il metodo isAperto() deve restituire true. Se la data del torneo è passata, deve restituire false.

MazzoTest

1. **testGeneraCodice()**
   - Verifica che il codice del mazzo venga generato correttamente. Prima del test, si assicura che il mazzo non abbia un codice (verificando che getCodice() restituisca null). Successivamente, si chiama il metodo setCodice() per generare il codice, e si verifica che il codice del mazzo non sia più null, confermando che è stato generato correttamente.

FormatoTorneoTest

1. **testLoadTipiMazzoMagic()**
   - Verifica che i tipi di mazzo vengano caricati correttamente per il formato di torneo "Magic: The Gathering". Il test chiama il metodo loadTipiMazzo() per caricare i tipi di mazzo, e successivamente verifica che la mappa dei tipi di mazzo contenga un solo tipo (con chiave 1) e che il nome di questo tipo di mazzo sia "Mazzo Pauper.

2. **testLoadTipiMazzoPokemon()**
   - Verifica che i tipi di mazzo vengano caricati correttamente per il formato di torneo "Pokémon". Il test chiama il metodo loadTipiMazzo() per caricare i tipi di mazzo, e successivamente verifica che la mappa dei tipi di mazzo contenga un solo tipo (con chiave 1) e che il nome di questo tipo di mazzo sia "Mazzo monotype".

3. **testLoadTipiMazzoYuGiOh()**
   - Verifica che i tipi di mazzo vengano caricati correttamente per il formato di torneo "Yu-Gi-Oh!". Il test chiama il metodo loadTipiMazzo() per caricare i tipi di mazzo, e successivamente verifica che la mappa dei tipi di mazzo

contenga tre tipi (con chiavi 1, 2, e 3) e che i nomi corrispondano ai tipi di mazzo previsti: "Main deck", "Extra deck" e "Side deck".