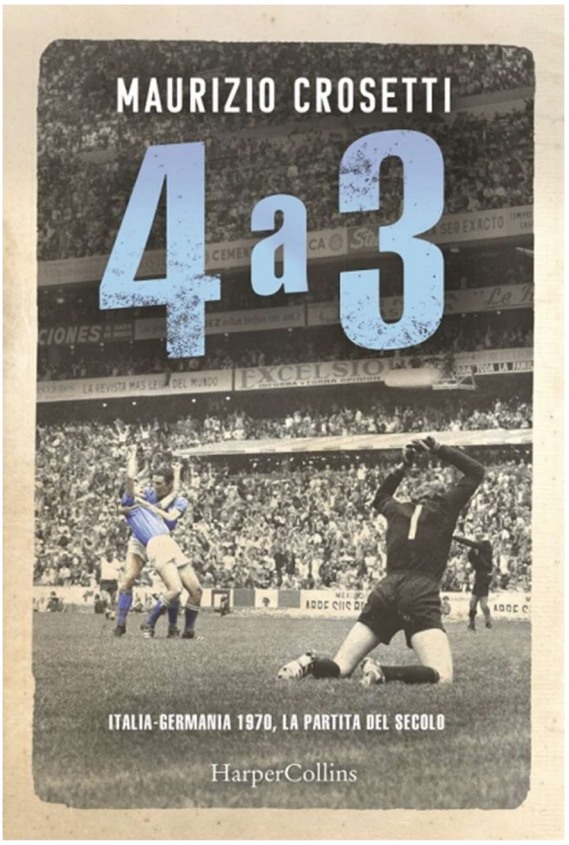
Corso di laurea in Informatica
Dipartimento di Informatica - Università di Pisa
Progetto Anno 2021/2022 – Appello Straordinario
IS: Consegna entro il 27-10-2022 ore 10:00

Introduzione comune per i corsi di IS e BD

Il torneo di calcetto
(in attesa della partita del secolo XXI)



Il calcetto, o calcio a 5, è divenuto molto popolare negli ultimi 30 anni con l'espansione piú forte avvenuta negli anni 90. Sono nati campi in erba sintetica un po' in tutta Italia. E' fondamentalmente il gioco del calcio che si disputa su un campo piú piccolo. A seconda delle dimensioni del terreno di gioco si può giocare in 5, 6 o 7 persone per squadra.

Serve realizzare un sistema software di supporto all'organizzazione e gestione di un torneo di calcetto, per fissare date e campi, avvisare i giocatori, lasciare messaggi ed aggiornare risultati e classifiche.

Occorre stabilire quale sarà il campo da gioco, o piú campi da gioco. Affittare un campo in erba sintetica in notturna costa circa 70-90 euro (in erba naturale 90-100 euro), compresi spogliatoi per doccia, in diurna un po' di meno. Per risparmiare si potrebbe giocare il sabato o la domenica mattina. Ad ogni giocatore quindi è richiesta una quota per finanziare anche l'acquisto di altro materiale, occorrono:

- 25 pettorine, o multipli se si giocano più partite il parallelo
- 1 pallone o multipli se si giocano più partite il parallelo: si gioca in quasi tutti i campi con il pallone numero 4 (per le regole FIFA deve pesare tra 400 e 440 grammi e se lasciato cadere da 2 metri di altezza il rimbalzo deve essere compreso tra 50 e 65 cm)
- arbitri: per tornei informali potreste farne anche a meno

Il sistema deve permettere agli organizzatori di:

- personalizzare il torneo scegliendo se farlo ad eliminazione diretta o a girone unico o andata e ritorno;

- aggiornare i risultati;
- fare le convocazioni;
- creare una bacheca dove tutti gli interessati possono lasciare informazioni sulle prossime partite da disputare.

Il sistema deve permettere ai partecipanti di:

- registrarsi e gestire il proprio profilo;
- leggere i messaggi ricevuti dagli organizzatori;
- registrarsi ad uno o più tornei;
- lasciare i commenti su una partita appena disputata.

Il sistema permette non solo di gestire il torneo ma anche di trovare nuovi giocatori per allargare la nostra rosa.
Se invece siete appassionati di calcetto e ogni tanto avete voglia di dare 4 calci al pallone potete dare un'occhiata agli annunci in bacheca. Potete fare una ricerca in base alla città e zone con la quota per partecipare alla partita. Cliccate e raggiungete il campo per giocare.

Progetto di Ingegneria del Software

Domanda1. Riscrivere i requisiti del sistema *Il torneo di calcetto* eliminando ogni ambiguità o altro difetto evidente (evidenziando le parti modificate o aggiunte) e usando un formato standard. Riscrivere 3 tre requisiti anche usando il modello delle user stories.

Tornei

1. L'organizzatore deve essere in grado di creare un torneo
2. L'organizzatore deve essere in grado di scegliere il tipo di torneo (eliminazione diretta, girone con partite uniche tra coppie di squadre o andata e ritorno)

3. L'organizzatore deve essere in grado di fissare date e campi di gioco di un torneo
4. L'organizzatore deve essere in grado di fissare una quota di partecipazione a un torneo
5. L'organizzatore deve essere in grado di convocare i giocatori
6. L'organizzatore deve essere in grado di inviare messaggi e-mail agli iscritti al torneo
7. L'utente deve essere in grado di registrarsi a uno o più tornei
8. L'organizzatore deve essere in grado di aggiornare i risultati delle partite
9. Il sistema in seguito all'inserimento di un risultato deve calcolare il nuovo stato del torneo

Bacheca

10. L'organizzatore deve essere in grado di creare una bacheca
11. Il sistema deve permettere a utenti e organizzatori di creare degli annunci sulla bacheca
12. Il sistema deve permettere a utenti e organizzatori di lasciare sulla bacheca informazioni sulle prossime partite da disputare.
13. Il sistema deve permettere la ricerca di annunci e informazioni in bacheca in base a città, zona e quota di partecipazione
14. L'utente deve essere in grado di lasciare un commento in bacheca su una partita appena disputata

Utenti

15. Il sistema deve permettere a nuovi utenti di registrarsi
16. L'utente deve essere in grado di aggiornare il proprio profilo

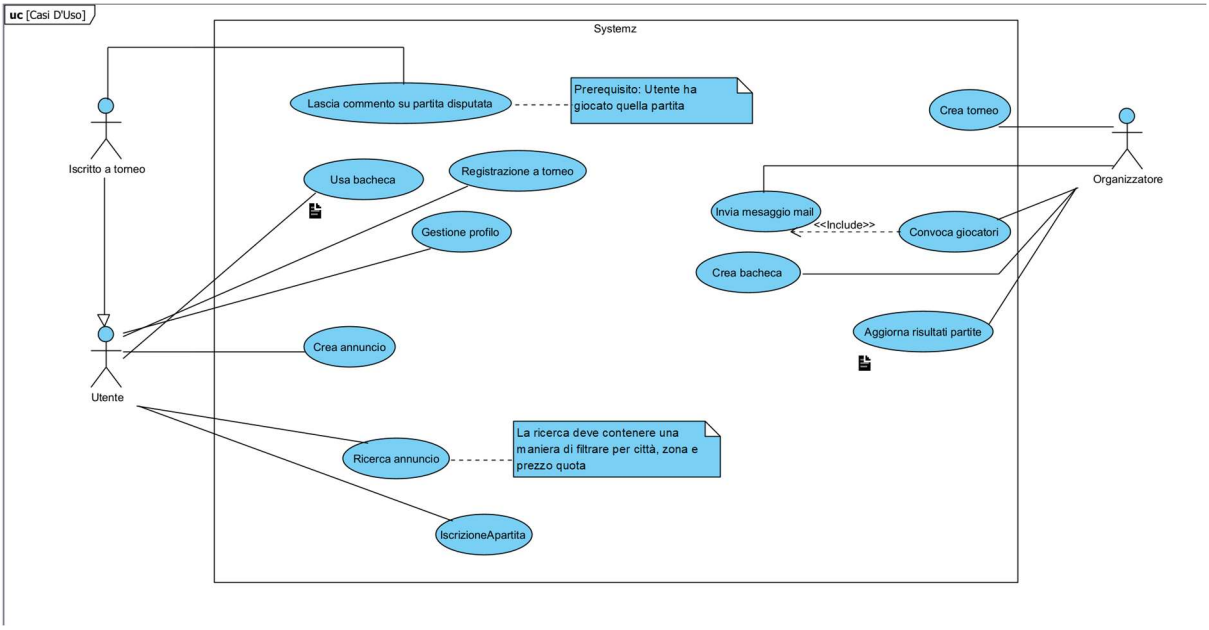
User Stories

Come utente, devo avere la possibilità di inserire un commento su una partita giocata al fine di interagire con gli altri utenti della piattaforma.

In veste di organizzatore devo essere in grado di inviare messaggi e-mail agli iscritti al torneo, al fine di fornire loro notifiche e informazioni pertinenti relative al torneo stesso.

In qualità di appassionato di calcio devo essere in grado di ricercare annunci per trovare una partita da giocare

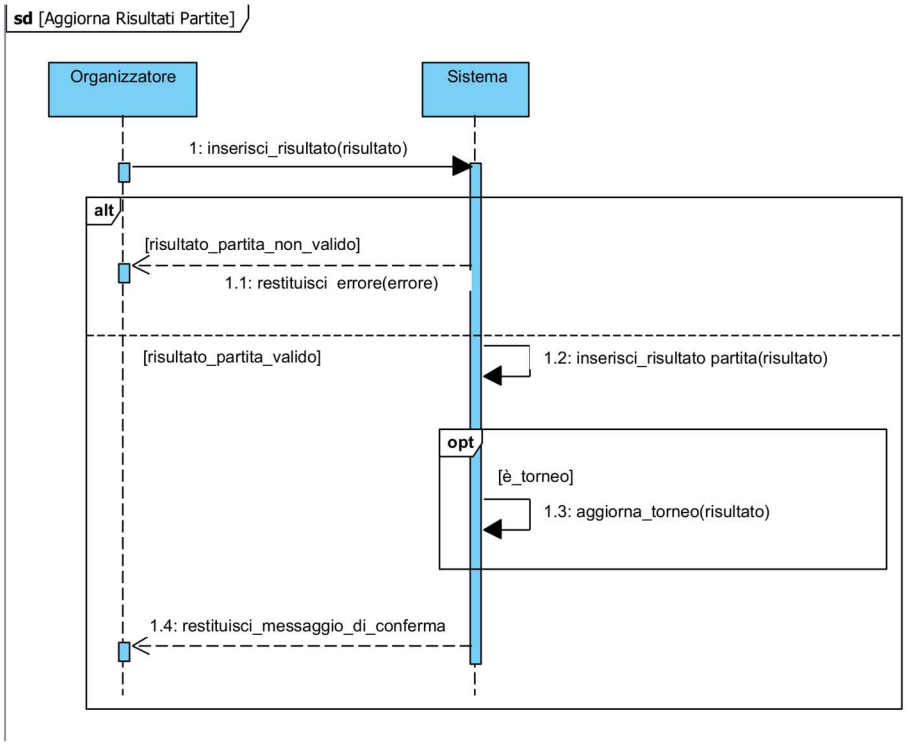
Domanda 2. Descrivere con un diagramma UML tutti i casi d'uso del Sistema. Per uno di essi, non banale, dare la narrativa.



Caso d'uso: Usa bacheca
Breve descrizione: Un iscritto al torneo vuole leggere o scrivere sulla bacheca
Precondizione: La bacheca deve essere creata
PostCondizione: (Se ci sono post presenti in bacheca sono stati mostrati i i post in bacheca) AND (se un utente ha scritto un post in bacheca il post
è pubblicato) AND (se un utente ha fatto una ricerca in bacheca ha ottenuto gli annunci che soddisfano le chiavi di ricerca)
Sequenza principale degli eventi:
1.L'utente accede alla bacheca
2. Se (ci sono post presenti in bacheca)
    2.1. Il sistema mostra i post in bacheca
3. Se(utente vuole scrivere su bacheca)
    3.1. Utente scrive un post
    3.2. Il sistema aggiunge il post alla bacheca
4. Se(utente vuole cercare annunci in bacheca)
    4.1 L'Utente indica le chiavi di ricerca
    4.2 Il sistema mostra gli annunci che soddisfano le chiavi di ricerca

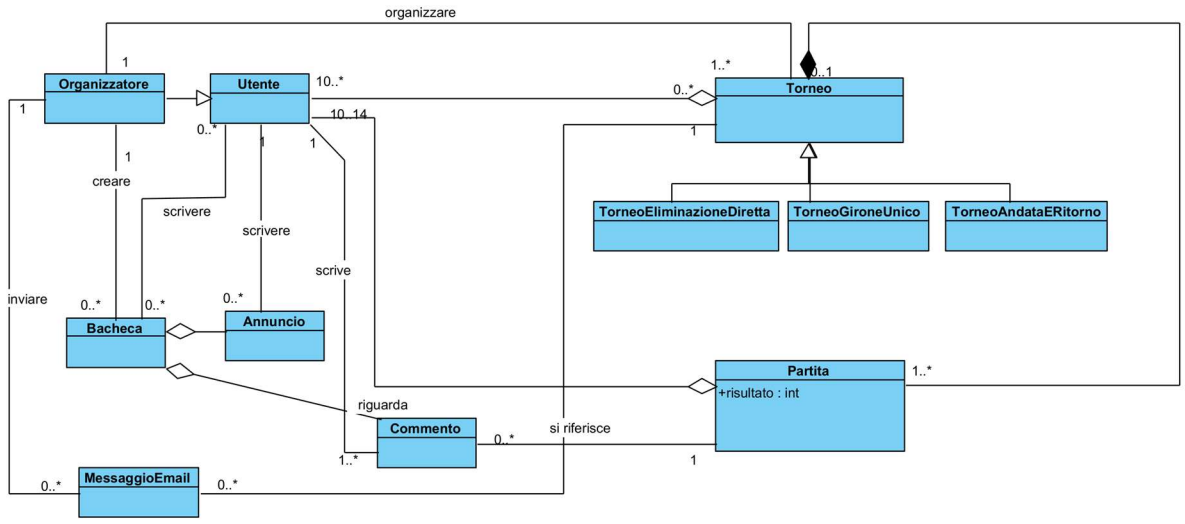
Caso d'uso: Aggiorna risultati partite
Breve descrizione: Un organizzatore vuole inserire il risultato di una partita
Precondizione: La partita deve essere terminata
Postcondizione: (Risultato aggiunto e (se la partita fa parte di un torneo, torneo aggiornato)) o problema segnalato
Sequenza di eventi:
1. L'organizzatore inserisce il risultato della partita
2 Il sistema controlla che il risultato della partita sia valido
3. Il sistema controlla se la partita fa parte di un torneo
4. Se (risulltato partita valido)
    4.1. Il sistema inserisce il risultato della partita
5. Altrimenti il Sistema segnala il problema all'organizzatore
6. Se(la partita fa parte di un torneo)
    6.1 Il sistema aggiorna lo stato del torneo in base al risultato della partita

Domanda 3. Dare un diagramma di sequenza che descriva il caso d'uso di cui si è data la narrativa.

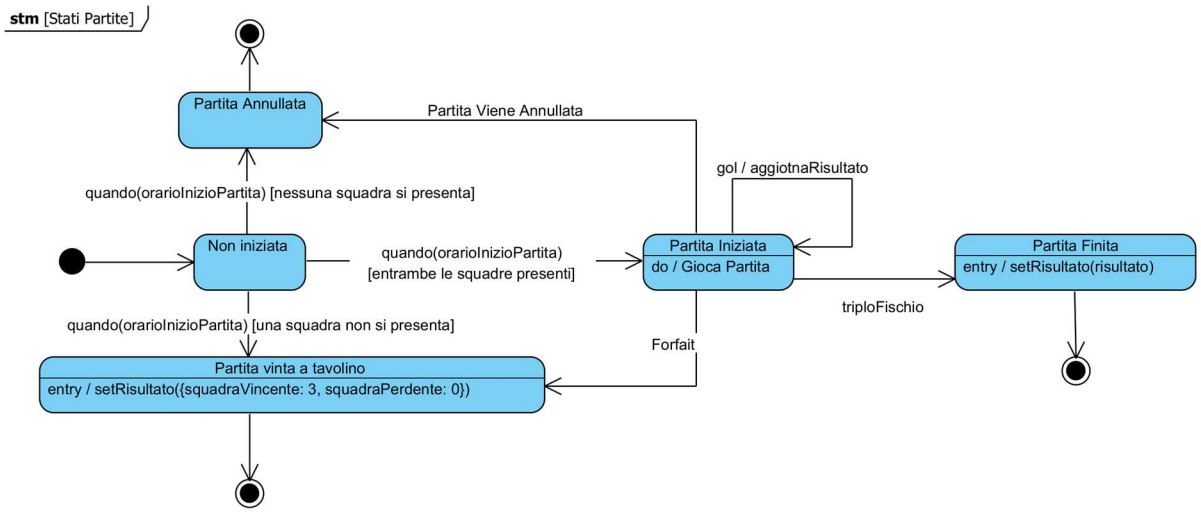


Domanda 4. Dare un diagramma delle classi che descriva gli elementi significativi del dominio del sistema.

Diagramma parziale che non considera pettorine, palloni etc.



Domanda 5. Dare un diagramma di macchina a stati che modelli gli stati in cui può trovarsi una partita, considerando che è possibile venga interrotta e venga data vittoria a tavolino a una delle squadre. Lo stesso succede se una squadra non si presenta in tempo.



Domanda 6. Definire una possibile architettura (dare sia la vista C&C che di dislocazione) per realizzare il sistema in oggetto.

Domanda 7. Presso alcuni campi esiste un negozio che vende articoli quali scarpette, pettorine, palloni etc. Per calcolare il prezzo totale di un acquisto è stato scritto un metodo che riceve in input un array con i codici degli articoli acquistati e restituisce il totale della spesa. Il metodo tiene conto del fatto che alcuni articoli possano essere messi in offerta, con offerta: 50% di sconto sul secondo articolo:

```
// db denota l'oggetto che gestisce le interazioni col DB

public static int calcola (int [] articoliComprati) {
   int totale = 0;
   for(int i=0; i < articoliComprati.length; i++) {
        int articolo = articoliComprati[i];
        if(articolo != -1 ){
               int prezzo = db.getPrezzo(articolo);
               if(db.offerta(articolo)) {
                int count = 1;
                 for(int j=i+1; j<articoliComprati.length; j++){
                   if(articolo == articoliComprati[j]) {
                        count++; articoliComprati[j]=-1;}
                 }
               prezzo = prezzo*0.75;
               }
        totale = totale + prezzo;
        }
   }
return totale;
}
```

a) Fornire alcuni casi di prova per verificare se il metodo si comporta correttamente per quanto riguarda la gestione delle offerte. Per la progettazione del caso di prova si usi il criterio funzionale basato sulla partizione dei dati in ingresso. Si assuma che i codici siano interi, e che, nell'ambiente di test, tutti i prodotti in vendita con codice multiplo di 3 siano in offerta e che il prezzo di un articolo sia pari a 10 volte il suo codice.

Caso di prova senza offerte:
Input: `[1, 2, 4, 5, 7]` (nessun articolo in offerta)
Previsto: Totale = 1 * 10 + 2 * 10 + 4 * 10 + 5 * 10 + 7 * 10 = 100

Caso di prova con offerte ma senza articoli ripetuti:
Input: `[3, 6, 9, 12]` (articoli con codice multiplo di 3 sono in offerta)
Previsto: Totale = 3 * 10 + 6 * 10 + 9 * 10 + 12 * 10 = …

Caso di prova con offerte su alcuni articoli ma senza articoli ripetuti:
Input: `[3, 2, 9, 5]` (articoli con codice multiplo di 3 sono in offerta)
Previsto: Totale = 3 * 10 + 2 * 10 + 9 * 10 + 5 * 10 = …

Caso di prova senza articoli:
Input: `[]` (nessun articolo)
Previsto: Totale = 0

Caso di prova con articoli duplicati senza offerte:
Input: `[1, 1, 2, 2, 4, 4]` (nessun articolo in offerta)
Previsto: Totale = 1 * 10 + 1 * 10 + 2 * 10 + 2 * 10 + 4 * 10 + 4 * 10 = 140

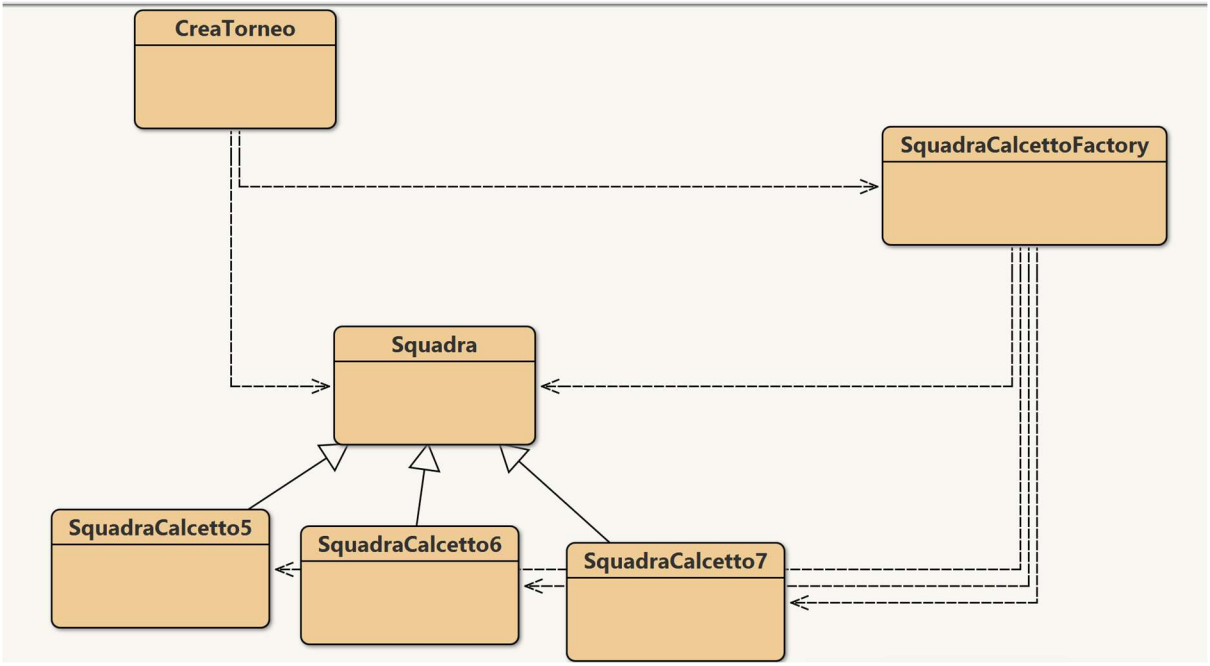
Caso di prova con articoli duplicati con offerte:
Input: `[3, 3, 6, 6, 9, 9]` (articoli con codice multiplo di 3 sono in offerta)
Previsto: Totale = 3 * 10 + 3 * 10 * 0.5 + 6 * 10 + 6 * 10 * 0.5 + 9 * 10 + 9 * 10 * 0.5 = 97.5

Caso di prova generico:
Input: `[2, 3, 3, 5, 6, 9, 9]` (articoli con codice multiplo di 3 sono in offerta)
Previsto: Totale = 2 * 10 + 3 * 10 + 3 * 10 * 0.5 + 5 * 10 + 6 * 10 + 9 * 10 + 9 * 10 * 0.5 = ….

b) Verificare se l'esecuzione del codice con i casi di prova dati in a) evidenzia un malfunzionamento.
TBD

c) Costruire il grafo di flusso del metodo usando la notazione dei diagrammi di attività UML e definire una proof obligation per copertura dei 3-cicli (fino a 3 iterazioni).
TBD

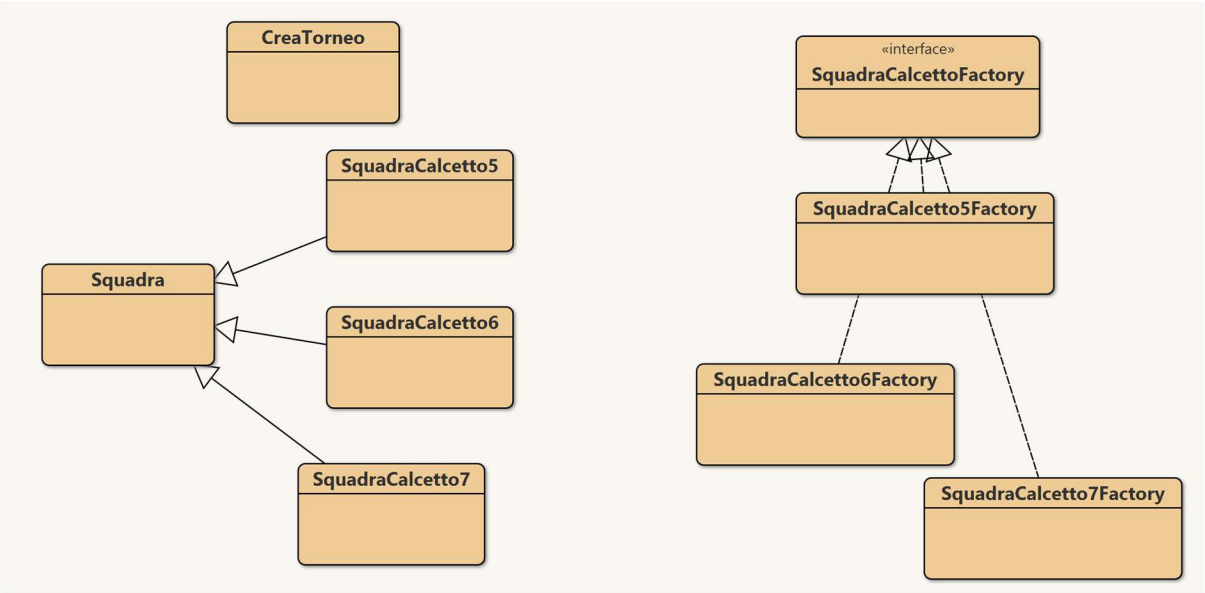
Uso di Factories per la creazione di Tornei.




```java
import java.util.ArrayList;
import java.util.List;

// Classe che gestisce la creazione di un torneo
public class CreaTorneo {

    private SquadraCalcettoFactory factory;

    public CreaTorneo(SquadraCalcettoFactory factory) {
        this.factory = factory;
    }

    // Metodo per inizializzare un torneo con n squadre da x giocatori ciascuna
    public List<Squadra> inizializzaTorneo(int numeroSquadre, int numeroGiocatori) {
        List<Squadra> squadre = new ArrayList<>();
        for (int i = 0; i < numeroSquadre; i++) {
            Squadra squadra = factory.creaSquadra(numeroGiocatori);
            squadre.add(squadra);
        }
        return squadre;
    }
}
```

```java
// Concrete Factory per il torneo di calcetto
class SquadraCalcettoFactory {
    Squadra creaSquadra(int numeroGiocatori) {
        if (numeroGiocatori == 5) {
            return new SquadraCalcetto5();
        } else if (numeroGiocatori == 6) {
            return new SquadraCalcetto6();
        } else if (numeroGiocatori == 7) {
            return new SquadraCalcetto7();
        } else {
            throw new IllegalArgumentException("Numero di giocatori non supportato");
        }
    }
}
```

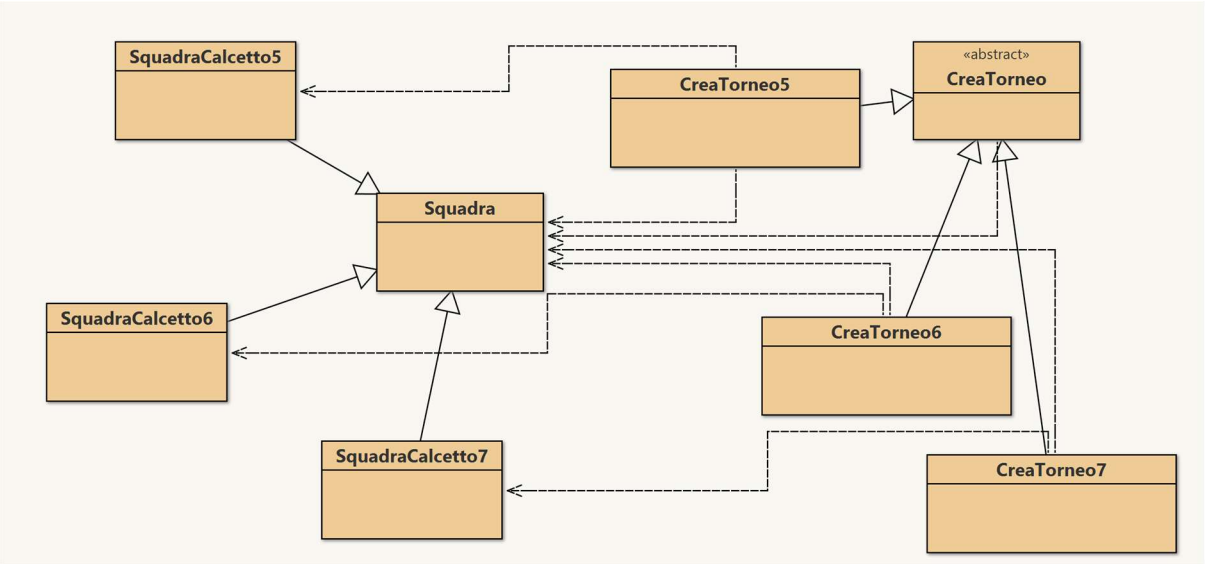
USANDO ABSTRACT FACTORY




```java
public class CreaTorneo {

    private SquadraCalcettoFactory factory;

    // Metodo per inizializzare un torneo con n squadre dicendo quale Factory usare
    public List<Squadra> inizializzaTorneo(int numeroSquadre, SquadraCalcettoFactory factory) {
        this.factory = factory
        List<Squadra> squadre = new ArrayList<>();
        for (int i = 0; i < numeroSquadre; i++) {
            Squadra squadra = factory.creaSquadra();
            squadre.add(squadra);
        }
        return squadre;
    }

```

```java
// Abstract Factory per il torneo di calcetto
public interface SquadraCalcettoFactory {
    Squadra creaSquadra();
}


// Concrete Factory per il torneo di calcetto con 5 giocatori
public class SquadraCalcetto5Factory implements SquadraCalcettoFactory {
    public Squadra creaSquadra() {
        return new SquadraCalcetto5();
    }
}

// Concrete Factory per il torneo di calcetto con 6 giocatori
public class SquadraCalcetto6Factory implements SquadraCalcettoFactory {
    public Squadra creaSquadra() {
        return new SquadraCalcetto6();
    }
}

// Concrete Factory per il torneo di calcetto con 7 giocatori
public class SquadraCalcetto7Factory implements SquadraCalcettoFactory {
    public Squadra creaSquadra() {
        return new SquadraCalcetto7();
    }
}
```

USANDO FACTORY METHOD




```java
//Creator
public abstract class CreaTorneo {

    public abstract Squadra creaSquadra();

    // Metodo per inizializzare un torneo con n squadre
    public List<Squadra> inizializzaTorneo(int numeroSquadre) {
        List<Squadra> squadre = new ArrayList<>();
        for (int i = 0; i < numeroSquadre; i++) {
            Squadra squadra = creaSquadra();
            squadre.add(squadra);
        }
        return squadre;
    }
}

// Concrete Creators:
public class CreaTorneo5 extends CreaTorneo{
    public Squadra creaSquadra() {
        return new SquadraCalcetto5();
    }
}
public class CreaTorneo6 extends CreaTorneo{
    public Squadra creaSquadra() {
        return new SquadraCalcetto6();
    }
}
public class CreaTorneo7 extends CreaTorneo{
    public Squadra creaSquadra() {
        return new SquadraCalcetto7();
    }
}
```

CONCRETE FACTORY

```
/ Classe che gestisce la creazione di un torneo
public class CreaTorneo {

    private SquadraCalcettoFactory factory;
```

```
    public CreaTorneo(SquadraCalcettoFactory factory) {
        this.factory = factory;
    }

    // Metodo per inizializzare un torneo con n squadre da x giocatori
ciascuna
    public List<Squadra> inizializzaTorneo(int numeroSquadre, int
numeroGiocatori) {
        List<Squadra> squadre = new ArrayList<>();
        for (int i = 0; i < numeroSquadre; i++) {
            Squadra squadra = factory.creaSquadra(numeroGiocatori);
            squadre.add(squadra);
        }
        return squadre;
    }


// Concrete Factory per il torneo di calcetto
class SquadraCalcettoFactory {
    Squadra creaSquadra(int numeroGiocatori) {
        if (numeroGiocatori == 5) {
            return new SquadraCalcetto5();
        } else if (numeroGiocatori == 6) {
            return new SquadraCalcetto6();
        } else if (numeroGiocatori == 7) {
            return new SquadraCalcetto7();
        } else {
            throw new IllegalArgumentException("Numero di giocatori
non supportato");
        }
    }
}
```

Abstract Factory:

```
public class CreaTorneo {

    private SquadraCalcettoFactory factory;

    // Metodo per inizializzare un torneo con n squadre dicendo
quale Factory usare
    public List<Squadra> inizializzaTorneo(int numeroSquadre,
SquadraCalcettoFactory factory) {
        this.factory = factory
        List<Squadra> squadre = new ArrayList<>();
        for (int i = 0; i < numeroSquadre; i++) {
            Squadra squadra = factory.creaSquadra();
            squadre.add(squadra);
        }
```

```
        return squadre;
    }

// Abstract Factory per il torneo di calcetto
public interface SquadraFactory {
    Squadra creaSquadra();
}


// Concrete Factory per il torneo di calcetto con 5 giocatori
public class SquadraCalcetto5Factory implements SquadraFactory {
    public Squadra creaSquadra() {
        return new SquadraCalcetto5();
    }
}

// Concrete Factory per il torneo di calcetto con 6 giocatori
public class SquadraCalcetto6Factory implements SquadraFactory {
    public Squadra creaSquadra() {
        return new SquadraCalcetto6();
    }
}

// Concrete Factory per il torneo di calcetto con 7 giocatori
public class SquadraCalcetto7Factory implements SquadraFactory {
    public Squadra creaSquadra() {
        return new SquadraCalcetto7();
    }
}
```

Factory Method

```
//Creator
public abstract class CreaTorneo {

    public abstract Squadra creaSquadra();

    // Metodo per inizializzare un torneo con n squadre
    public List<Squadra> inizializzaTorneo(int numeroSquadre) {
        List<Squadra> squadre = new ArrayList<>();
        for (int i = 0; i < numeroSquadre; i++) {
            Squadra squadra = creaSquadra();
            squadre.add(squadra);
        }
        return squadre;
    }
}

// Concrete Creators:
public class CreaTorneo5 extends CreaTorneo{
    public Squadra creaSquadra() {
        return new SquadraCalcetto5();
```

```
    }
}
public class CreaTorneo6 extends CreaTorneo{
    public Squadra creaSquadra() {
        return new SquadraCalcetto6();
    }
}
public class CreaTorneo7 extends CreaTorneo{
    public Squadra creaSquadra() {
        return new SquadraCalcetto7();
    }
}
```