

# UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II  
CORSO DI LAUREA TRIENNALE IN INFORMATICA

## Documentazione Hackathon Versione 1

Mario Majorano                    N86005035  
Luca Sanselmo                    N86005147

Anno Accademico 2024/2025

---

# Indice

<b>1 Modello di Dominio</b>	<b>2</b>
1.1 Introduzione . . . . .	2
1.2 Requisiti . . . . .	2
1.3 Analisi Dei Requisiti . . . . .	3
1.3.1 Requisiti Informativi . . . . .	3
1.3.2 Requisiti Funzionali . . . . .	4
<b>2 Classi, Associazioni e Gerarchie</b>	<b>5</b>
2.1 Classi, Attributi e/o Responsabilità . . . . .	5
2.2 Gerarchie tra Classi . . . . .	6
2.3 Associazioni tra Classi - Note . . . . .	6
2.3.1 Classi Associative . . . . .	7
<b>3 Dizionari</b>	<b>7</b>
3.1 Dizionario delle Classi . . . . .	7
3.2 Dizionario delle Associazioni . . . . .	8
3.2.1 Dizionario Classi Associate . . . . .	8
<b>4 Class Diagram</b>	<b>9</b>
<b>5 Implementazione - Packages</b>	<b>10</b>
5.1 Model . . . . .	10
5.2 Package Diagram . . . . .	11
5.3 Note . . . . .	12
<b>6 Repository GitHub</b>	<b>12</b>

# 1 Modello di Dominio

## 1.1 Introduzione

La presente documentazione ha lo scopo di (1) descrivere l'analisi effettuata per la risoluzione del Modello di Dominio del Problema proposto dal cliente, di (2) portare il lettore alla comprensione delle scelte implementative adottate per la progettazione concettuale (3) ed infine quello di rappresentare gli elementi concreti descritti dal contesto del mondo reale trattato tramite la definizione di Classi, caratterizzate dai relativi Attributi e Responsabilità.

Tale progettazione concettuale include esclusivamente gli attributi e le responsabilità descritte esplicitamente dal cliente.

## 1.2 Requisiti

I requisiti sono specificati all'interno della seguente traccia di esercizio:

*Un hackathon, ovvero una "maratona di hacking", è un evento durante il quale team di partecipanti si sfidano per progettare e implementare nuove soluzioni basate su una certa tecnologia o mirate a un certo ambito applicativo.*

*Ogni hackathon ha un titolo identificativo, si svolge in una certa sede e in un certo intervallo di tempo (solitamente 2 giorni) e ha un organizzatore specifico (registrato alla piattaforma). L'organizzatore seleziona un gruppo di giudici (selezionati tra gli utenti della piattaforma, invitandoli). Infine, l'organizzatore apre le registrazioni, che si chiuderanno 2 giorni prima dell'evento. Ogni evento avrà un numero massimo di iscritti e una dimensione massima del team.*

*Durante il periodo di registrazione, gli utenti possono registrarsi per l'Hackathon di loro scelta (eventualmente registrandosi sulla piattaforma se non lo hanno già fatto). Una volta iscritti, gli utenti possono formare team. I team diventano definitivi quando si chiudono le iscrizioni. All'inizio dell'hackathon, i giudici pubblicano una descrizione del problema da affrontare.*

*Durante l'hackathon, i team lavorano separatamente per risolvere il problema e devono caricare periodicamente gli aggiornamenti sui "progressi" sulla piattaforma come documento, che può essere esaminato e commentato dai giudici. Alla fine dell'hackathon, ogni giudice assegna un voto (da 0 a 10) a ciascun team e la piattaforma, dopo aver acquisito tutti i voti, pubblica le classifiche dei team.*

## 1.3 Analisi Dei Requisiti

### 1.3.1 Requisiti Informativi

Il sistema prevede la realizzazione di una piattaforma atta all'organizzazione ed alla gestione di una maratona di Hacking in cui team di partecipanti collaborano per sviluppare soluzioni innovative in un tempo limitato, solitamente 48 ore. Ogni hackathon è unico e prevede una serie di caratteristiche ben definite che ne regolano l'organizzazione:

- Organizzazione dell' hackathon:

L'hackathon viene creato dalla figura dell'**Organizzatore**, un utente registrato sulla piattaforma, che ne definisce i dettagli fondamentali: un *titolo* identificativo, la *sede* in cui si svolgerà, la *durata* (tipicamente due giorni), e parametri operativi come il numero massimo di partecipanti e la *dimensione massima dei team*.

- Selezione dei Giudici:

Una volta configurato l'evento, l'**Organizzatore** seleziona un gruppo di **giudici** tra gli utenti già presenti nella piattaforma, invitandoli a valutare i progetti presentati dai *partecipanti*.

- Registrazione e formazione dei team:

Le iscrizioni all'**hackathon** vengono aperte dall'organizzatore e rimangono disponibili fino a 48 ore prima dell'inizio dell'evento, momento in cui i team formati diventano definitivi. In questo periodo, gli utenti possono registrarsi singolarmente o formare team. Se un utente non è ancora registrato sulla piattaforma, può creare un account in fase di iscrizione.

- Svolgimento dell'hackathon

All'inizio della competizione, i giudici pubblicano la *descrizione del problema* da risolvere, che servirà come base per lo sviluppo dei progetti. Durante l'evento, i team lavorano in modo indipendente ma sono tenuti a caricare periodicamente aggiornamenti sui loro progressi sotto forma di documenti successivamente valutati dai giudici.

- Valutazione e classifica finale

Al termine dell'hackathon, ogni giudice assegna un *voto* a ciascun Team in base alla qualità della soluzione proposta. La piattaforma raccoglie automaticamente tutti i voti e pubblica una classifica finale resa disponibile a tutti i partecipanti.

#### Vincoli individuati:

Per garantire il corretto funzionamento della competizione, il sistema impone le seguenti regole:

- I **giudici** non possono prendere parte alla competizione in qualità di partecipanti
- Gli **organizzatori** non possono prendere parte alla competizione in qualità di partecipanti
- Un **team** può essere formato anche da un solo partecipante
- Un gruppo di **giudici**, per essere considerato valido, deve essere composto da almeno 2 membri
- Un **hackathon**, per essere considerato valido, deve prevedere almeno 2 **team** partecipanti
- Il **voto** finale  $v \in [0, 10]$

$$\mathcal{G} = \{g_1, \dots, g_k\} \quad |\mathcal{G}| \geq 2 \quad (\text{Giudici})$$

$$\mathcal{P} = \{p_1, \dots, p_m\} \quad (\text{Partecipanti})$$

$$\mathcal{O} = \{o_1, \dots, o_n\} \quad |\mathcal{O}| = 1 \quad (\text{Organizzatore, unico per ogni Hackathon})$$

$$\mathcal{T} = \{t_1, \dots, t_z \mid |t| \geq 2\} \quad (\text{Team, almeno 2 partecipanti})$$

$$\mathcal{H} = \langle \mathcal{P}, \mathcal{O}, \mathcal{G}, \mathcal{T}, \text{Problema} \rangle \quad (\text{Istanza di Hackathon})$$

### 1.3.2 Requisiti Funzionali

Elenco delle funzionalità e dei rispettivi dati da memorizzare

- **Organizzatore:** \_\_\_\_\_
  - **Creazione hackathon**
    - Titolo identificativo (univoco)
    - Durata
    - DataInizio
    - DataFine
    - Sede
    - Numero massimo partecipanti
    - Dimensione massima team
  - **Selezione giudici** \_\_\_\_\_
    - Assegnazione di un gruppo di Giudici ad uno specifico hackathon
- **Partecipante:** \_\_\_\_\_
  - **Registrazione all'Hackathon**
  - **Creazione e/o Partecipazione ad un Team**
    - Nome del team
- **Team:** \_\_\_\_\_
  - **Pubblicazione Documento**
  - Contenuto testuale dei progressi del Team
- **Giudici:** \_\_\_\_\_
  - **Sistema di valutazione**
    - Commenti testuali ai Documenti pubblicati dai Partecipanti
  - **Pubblicazione del problema**
    - Descrizione del problema

## 2 Classi, Associazioni e Gerarchie

Tali oggetti del mondo reale appena descritti sono stati rappresentati tramite l'individuazione e la successiva definizione di:

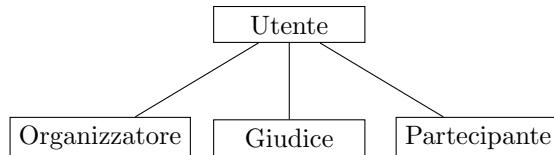
- Classi con relativi Attributi e Responsabilità
- Gerarchie tra Classi
- Associazioni tra Classi

### 2.1 Classi, Attributi e/o Responsabilità

- **Utente** \_\_\_\_\_
  - \* **Partecipante**
    - SingUpHackathon
    - createTeam
    - joinTeam
  - \* **Giudice**
    - publishProblem
    - commentProgress
    - gradeTeam
  - \* **Organizzatore**
    - inviteJudge
    - openHackathon
- **Team** \_\_\_\_\_
  - publishProgress
- **Documento** \_\_\_\_\_
  - Titolo
  - Contenuto
  - Commento
- **Hackathon** \_\_\_\_\_
  - Titolo
  - Sede
  - Durata
  - DataInizio
  - DataFine
  - PeriodoIscrizioni
  - DataAperturaIscrizioni
  - DataChiusuraIscrizioni
  - MaxIscritti
  - MaxDimTeam
  - DescrizioneProblema
- **Votazione** \_\_\_\_\_
  - Voto
- **Registrazione** \_\_\_\_\_

## 2.2 Gerarchie tra Classi

Durante la progettazione concettuale è trasparsa la necessità di descrivere una specializzazione della Classe Utente, attraverso tre sottoclassi: Organizzatore, Giudice e Partecipante.



La Gerarchia individuata possiede i seguenti vincoli:

- **Vincolo di Disgiunzione:** Un'istanza può essere membro di al più di una sottoclasse
- **Vincolo di Completezza Parziale:** Ogni istanza nella superclasse NON DEVE obbligatoriamente essere membro di qualche sottoclasse nella specializzazione

Tali vincoli garantiscono il corretto funzionamento del sistema garantendo tutto ciò descritto nell'analisi requisiti precedente :

- Un utente è considerato *Partecipante* solamente se completa la registrazione ad un hackathon. Gli utenti generici non hanno automaticamente questo status.
- I Giudici hanno un ruolo esclusivo: possono valutare i progetti ma non possono iscriversi come partecipanti agli hackathon che giudicano.
- Analogamente, gli Organizzatori hanno accesso solo alle funzioni di gestione dell'evento e non possono partecipare alla competizione.

## 2.3 Associazioni tra Classi - Note

Per rendere il sistema il più possibile aderente al contesto del mondo reale descritto, le associazioni

- **Partecipazione** (verso la classe Team),
- **Competizione** (verso la classe Hackathon),
- **Organizzazione** (verso la classe Hackathon).

sono state progettate per consentire una gestione dello **storico degli eventi e delle interazioni tra le entità del sistema**. Nel contesto di una piattaforma per la gestione e la memorizzazione di molteplici hackathon, è stato ritenuto di notevole importanza mantenere uno storico che consenta di tracciare in modo completo e coerente il ciclo di vita di ogni oggetto. Tale approccio aggiunge alla piattaforma:

- una **tracciabilità temporale** delle attività
- la possibilità di effettuare **analisi retrospettive** e di generare report dettagliati.
- **scalabilità**: prepara il sistema a possibili **evoluzioni future**

Quindi, dal punto di vista pratico sarà possibile ottenere:

- per ogni Partecipante: a quali Team ha partecipato
- per ogni Team: a quali Hackathon ha partecipato
- per ogni Organizzatore: quali Hackathon ha organizzato

Tutto ciò appena esposto ha l'obiettivo di garantire **coerenza, completezza e scalabilità** alla progettazione del sistema.

### 2.3.1 Classi Associative

Le due classi associative individuate nel Class Diagram - *Registrazione* e *Votazione* - identificano una particolare occorrenza di associazione tra classi, è quindi affiorata la necessità di introdurre questo costrutto. In particolare:

- Per *Registrazione* si è esplicitata l'importanza della corretta connessione tra *Hackathon* e *Partecipante*, mirata a sottolineare che una istanza di *Registrazione* fa riferimento ad un singolo *Hackathon* e *Partecipante*, individuando così la coppia correttamente e rappresentando l'oggetto del mondo reale - la registrazione - effettuata da un utente.
- Per *Votazione*, la classe associativa presenta l'attributo *Voto*, riferito sia al *Team* ricevente che al corrispettivo *Giudice* che lo emette; questo permette quindi di individuare la coppia e contemporaneamente conservare il valore della votazione.

## 3 Dizionari

### 3.1 Dizionario delle Classi

Classe	Descrizione	Attributi	Responsabilità
Utente	Classe base per tutti gli utenti del sistema		
Partecipante	Utente registrato come partecipante		<ul style="list-style-type: none"> <li>– <i>singUpHackathon()</i></li> <li>– <i>createTeam()</i></li> <li>– <i>joinTeam()</i></li> </ul>
Giudice	Utente abilitato alla valutazione		<ul style="list-style-type: none"> <li>– <i>publishProblem()</i></li> <li>– <i>commentProblem()</i></li> <li>– <i>gradeTeam()</i></li> </ul>
Organizzatore	Utente che gestisce gli eventi		<ul style="list-style-type: none"> <li>– <i>inviteJudge()</i></li> <li>– <i>openHackathon()</i></li> </ul>
Team	Gruppo di partecipanti		<ul style="list-style-type: none"> <li>– <i>publishProgress()</i></li> </ul>
Documento	File prodotto durante l'hackathon	<ul style="list-style-type: none"> <li>– Titolo</li> <li>– Contenuto</li> <li>– Commento</li> </ul>	<ul style="list-style-type: none"> <li>– Registrare lo stato di avanzamento</li> </ul>
Hackathon	Competizione di programmazione	<ul style="list-style-type: none"> <li>– Titolo</li> <li>– Sede</li> <li>– Durata</li> <li>– DataInizio</li> <li>– DataFine</li> <li>– PeriodoIscrizioni</li> <li>– DataAperturaIscrizioni</li> <li>– DataChiusuraIscrizioni</li> <li>– MaxPartecipanti</li> <li>– DimMaxTeam</li> <li>– DescrizioneProblema</li> </ul>	

Tabella 1: Descrizione delle classi del sistema - Sono stati indicati unicamente gli attributi e le responsabilità indicate esplicitamente nel Dominio del Problema

### 3.2 Dizionario delle Associazioni

Associazione	Tipologia	Descrizione
Competizione	Molti-a-Molti	<ul style="list-style-type: none"> <li>– Un team Compete in un 1 o più Hackathon</li> <li>– In 1 Hackathon competono 2 o più Team</li> </ul>
Pubblicazione	Uno-a-Molti	<ul style="list-style-type: none"> <li>– Un Team pubblica nessuno o più Documenti</li> <li>– Un documento è pubblicato da 1 ed un solo Team</li> </ul>
Esaminazione	Molti-a-Molti	<ul style="list-style-type: none"> <li>– Un Giudice esamina nessuno o più documenti</li> <li>– Un Documento è esaminato da 2 o più Giudici</li> </ul>
Organizzazione	Uno-a-Molti	<ul style="list-style-type: none"> <li>– Un organizzatore organizza 1 o più Hackathon</li> <li>– Un Hackathon è organizzato da 1 Organizzatore</li> </ul>
Selezione	Uno-a-Molti	<ul style="list-style-type: none"> <li>– Un organizzatore seleziona 2 o più Giudici</li> <li>– Un Giudice è selezionato da uno ed un solo Organizzatore</li> </ul>
Partecipazione	Molti-a-Molti	<ul style="list-style-type: none"> <li>– Un partecipante partecipa ad 1 o più Team</li> <li>– Un Team possiede 1 o più Partecipanti</li> </ul>

Tabella 2: Descrizione delle associazioni tra classi del sistema

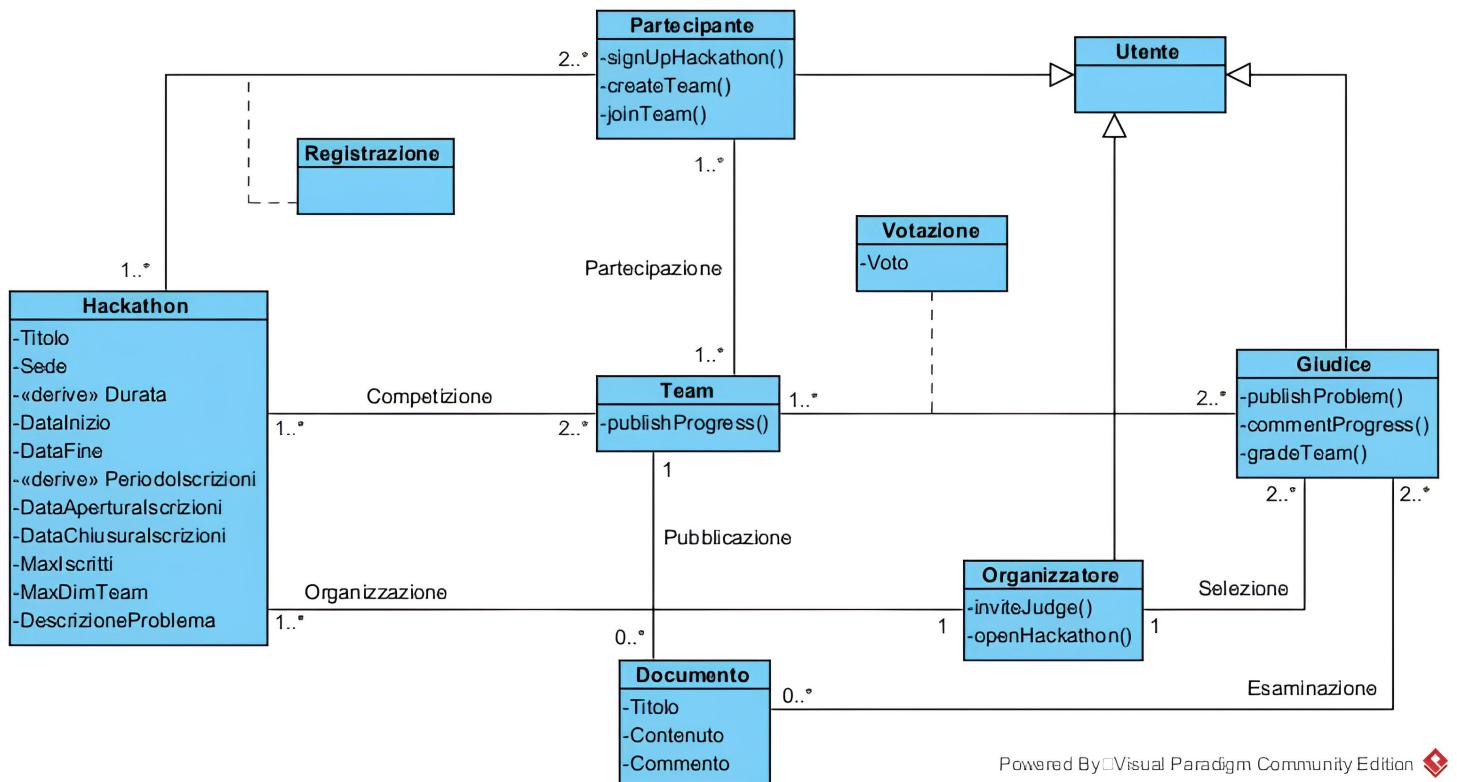
#### 3.2.1 Dizionario Classi Associative

Classe	Attributi	Descrizione
Registrazione	–	Azione con cui un utente partecipa a un hackathon
Votazione	Voto	Valutazione data da un giudice a un team

Tabella 3: Classi associative e relativi attributi

## 4 Class Diagram

Dall'analisi dei requisiti realizzata precedentemente, è stato ideato il seguente Class Diagram. Tale diagramma rappresenta il frutto della progettazione concettuale e descrive graficamente tramite lo standard UML le classi, le associazioni e le gerarchie succitate.



Powered By Visual Paradigm Community Edition

Figura 1: Diagramma delle Classi in UML

## 5 Implementazione - Packages

### 5.1 Model

Di seguito la rappresentazione del Package Model derivante dall'implementazione in linguaggio Java del Class Diagram (4) individuato illustrato precedentemente. Questo diagramma descrive esclusivamente il pacchetto *model* contenente le classi del dominio; gli altri pacchetti verranno illustrati successivamente.

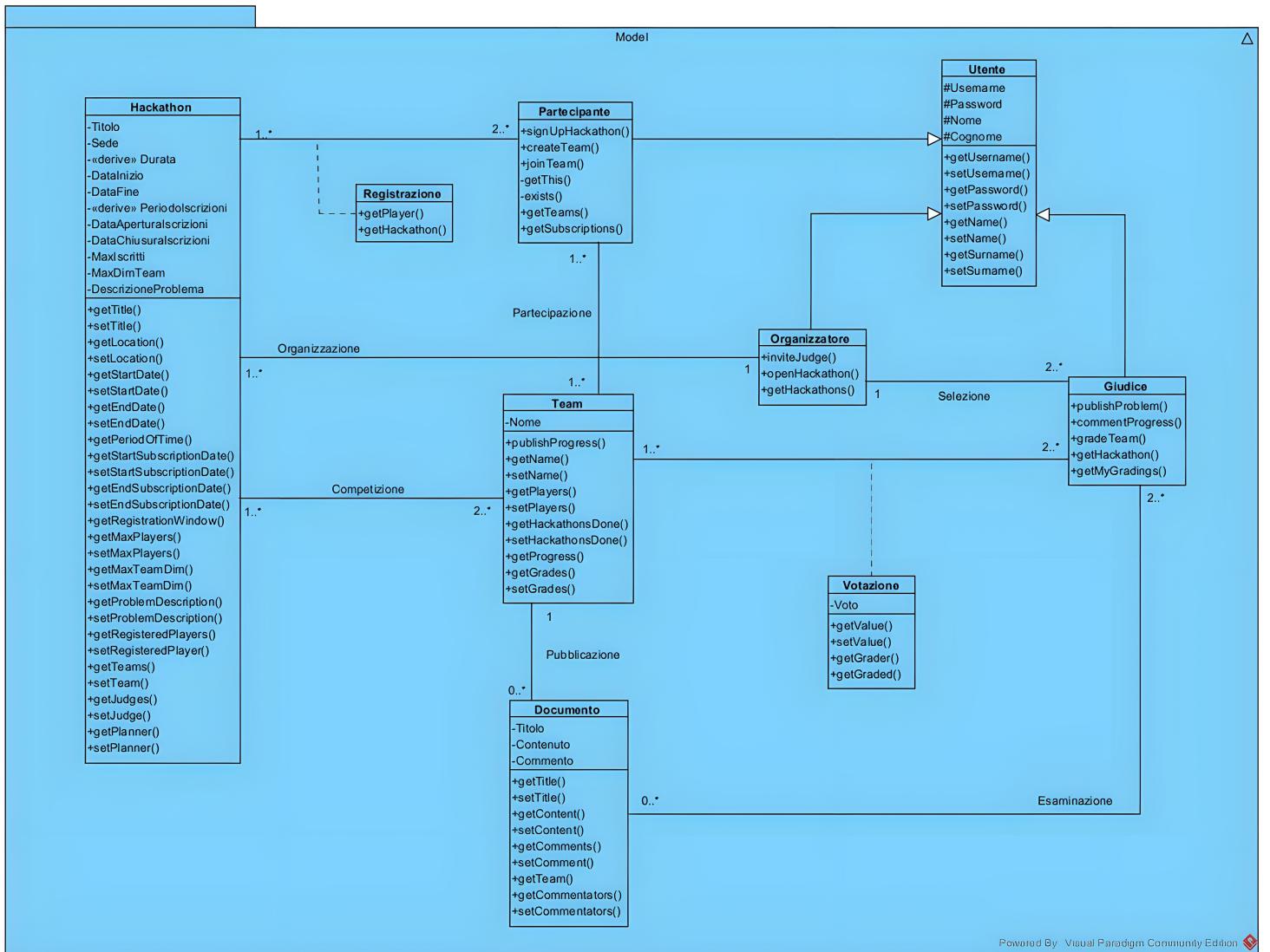


Figura 2: Package Model

## 5.2 Package Diagram

Il seguente diagramma di pacchetto illustra la struttura del codice Java per l'implementazione dell'interfaccia grafica. Il pacchetto *gui* descrive le funzionalità della Graphic User Interface (GUI) fornite all'utente. Il pacchetto *controller* descrive le funzioni necessarie per interagire con gli altri due pacchetti.

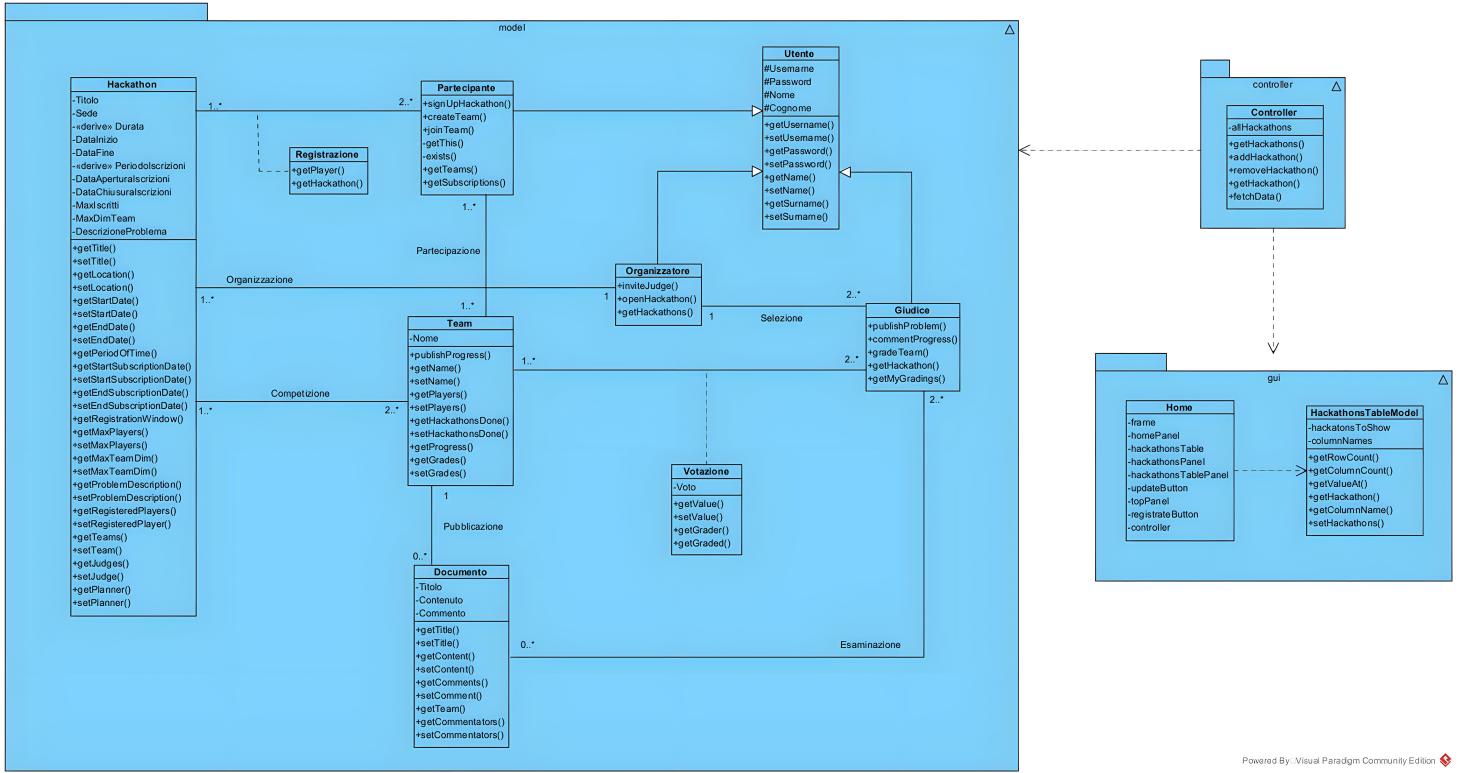


Figura 3: Package Diagram

### 5.3 Note

È importante sottolineare che questa implementazione è frutto di un Class Diagram non formalmente ristrutturato, quindi potrebbe essere soggetta a modifiche strutturali sostanziali per una migliore risoluzione del problema posto. In particolare sono evidenziate, qui e come commenti al codice, le seguenti possibili implementazioni:

- Partecipante
  - Potrebbe essere inserita una lista di Hackathon a cui il giocatore prende parte. Nonostante possa risultare un dato ridondante potrebbe tornare utile a fronte di una richiesta frequente di questa informazione.
- Hackathon
  - L'attributo *DataChiusuraIscrizioni* potrebbe essere derivabile da *DataInizio* sottraendovi 2 giorni. Questo però includerebbe un vincolo forte su queste date impedendo, per esempio, di sincronizzare la data di inizio Hackathon e la data di chiusura iscrizioni dello stesso.
  - È possibile separare la creazione dell'hackathon dall'apertura delle sue registrazioni. Questo sembra essere interessante in ottica di creazione di un torneo, ma potrebbe produrre hackathon mai iniziati. Da gestire correttamente per evitare memory leak.
- Registrazione
  - Il campo *Data* potrebbe essere introdotto per memorizzare la data di registrazione all'hackathon di riferimento; utile in caso di frequente richiesta di questo dato o a seguito di un ulteriore colloquio con il cliente.

In conclusione, alcune delle associazioni e/o classi associative possono essere implementate in diverso modo, ma al momento è sembrato opportuno procedere con questo tipo di approccio.

## 6 Repository GitHub

Il codice sorgente del progetto è pubblicamente disponibile su GitHub al seguente link:

[github.com/TrialShock26/Hackathon](https://github.com/TrialShock26/Hackathon)

La repository contiene:

- l'intero codice sorgente sviluppato
- la presente documentazione