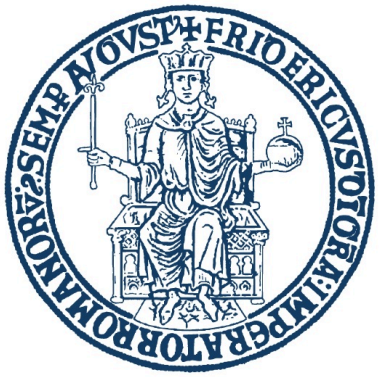


UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

PROGETTO DI OBJECT ORIENTATION A.A. 2024-2025
Primo Homework
24/04/2025

Petraccone Simone - N86005387
Picari Alessia - N86005131

0. Introduzione:

Traccia 2: Hackathon

Il progetto consiste nella realizzazione di una piattaforma per la gestione di un Hackathon.
La piattaforma permette agli utenti registrati di creare gare (dette Hackathon), formare team, caricare documenti di progetto e dare/ricevere valutazioni.

Dopo un'attenta lettura della traccia e dei requisiti del progetto, sono state individuate le classi principali necessarie, assegnando a ciascuna di esse i relativi attributi e metodi.

---

1. Individuazione delle classi e delle relative relazioni:
   La classe UtentePiattaforma rappresenta il concetto di utente generico registrato alla piattaforma di gestione degli Hackathon.
   È una superclasse da cui derivano in modo più specializzato, tre ulteriori classi (figlie):
   - Organizzatore: utente che crea e gestisce gli hackathon.
   - Giudice: utente che valuta i team assegnando voti.
   - Concorrente: utente che partecipa agli hackathon tramite un team.

   In questo modo, tutte le tipologie di utenti condividono le informazioni generiche di base.

   (Inoltre il tipo di associazione presente tra queste classi è una relazione di generalizzazione)

   - La classe UtentePiattaforma, presenta i seguenti

   Attributi:

   # nome: string
   # cognome: string
   # email: string

   I quali presentano una visibilità protected per i seguenti motivi:
   La visibilità protected consente a tutte le classi figlie (Organizzatore, Giudice e Concorrente) di accedere direttamente a questi attributi e di consentirne l'utilizzo dei metodi ereditati dalla superclasse.
   In questo modo, gli attributi non sono pubblici (quindi non esposti a tutte le classi) ma accessibili solo alle classi derivate (sottoclassi).

   NOTA.1:

   ➔ Per una questione di ordine e ridondanza, nei metodi mostrati in questa parte della relazione, non verranno inclusi né parametri né tipi di ritorno. Vedasi classi in java (collegamento alla repository su Github, situato a fine documento) per visualizzare i metodi per interi.

Metodi:

+ UtentePiattaforma() ; *metodo costruttore, utilizzato per inizializzare un nuovo utente appena creato*
+ getNome() ; *metodo utilizzato per visualizzare il valore dell'attributo nome dell'utente invocante la funzione*
+ setNome() ; *metodo utilizzato per impostare un nuovo valore all'attributo nome dell'utente invocante la funzione*
+ getCognome() ; *analogo al getNome()*
+ setCognome() ; *analogo al setNome()*
+ getEmail() ; *analogo al getNome()*
+ setEmail() ; *analogo al setNome()*
+ visualizzaClassifica() ; *metodo utilizzato per visualizzare una classifica di un determinato Hackathon*

Dopo aver ereditato gli attributi comuni dalla superclasse UtentePiattaforma, sono stati definiti nelle classi figlie alcuni metodi specifici.

La classe Organizzatore è responsabile della creazione e gestione degli eventi Hackathon e della convocazione dei giudici.

- La classe Organizzatore, presenta i seguenti

Metodi:

+ Organizzatore() ; *metodo costruttore, utilizzato per inizializzare un nuovo organizzatore appena creato*
+ creaHackathon() ; *permette di creare un nuovo Hackathon*
+ convocaGiudice() ; *permette di convocare un giudice per un Hackathon creato dall'organizzatore stesso*

Relazioni:

Organizzatore 1 → * Hackathon ; *un organizzatore può creare 0 o più Hackathon nel corso del tempo (non esiste un limite massimo)*

Organizzatore * → * Giudice ; *un organizzatore può convocare 0 o più giudici nel corso del tempo (sia per lo stesso, che per diversi Hackathon)*

La classe Giudice è responsabile della valutazione e supervisione dei team partecipanti agli eventi Hackathon.

- La classe Giudice, presenta i seguenti

Metodi:

+ Giudice() ; *metodo costruttore, utilizzato per inizializzare un nuovo giudice appena creato*

+ assegnaVoto() ; *permette al giudice di dare un punteggio (compreso tra 0 e 10) a un team che gli è stato precedentemente assegnato*

Relazioni:

Giudice * → * Organizzatore ; *un giudice può essere convocato da 0 o più giudici durante il corso del tempo (ogni convocazione riguarda un Hackathon diverso)*

Giudice * → * Hackathon ; *un giudice può supervisionare da 0 a più Hackathon (anche nello stesso momento)*

Giudice * → * Documento ; *un giudice può esaminare da 0 a più documenti*

Giudice * → * Team ; *un giudice può giudicare 0 o più team sia nello stesso Hackathon che in Hackathon diversi*

Il Giudice valuta i team assegnando loro un voto, la relazione diretta non è tra Giudice e Team, ma passa attraverso la classe associativa Voto:

La classe Voto è di tipo associativa, infatti essa rappresenta il voto che viene assegnato da un giudice ad un determinato team.

- La classe Voto, presenta i seguenti

Attributi:

- valore: int

Metodi:

+ Voto() ; *metodo costruttore, utilizzato per inizializzare un nuovo voto appena creato*
+ getValore() ; *metodo getter, analogo ai precedenti*
+ setValore() ; *metodo che permette al giudice di cambiare la propria valutazione precedentemente assegnata ad un team*

La classe Team, associata alla classe Giudice tramite la classe Voto, serve a rappresentare i gruppi di Concorrenti che partecipano agli Hackathon.

- La classe Team, presenta i seguenti

Attributi:

- nome: string
- pw: string (*Password di accesso al Team*)
- hackathon: Hackathon ; *serve per rappresentare la relazione Hackathon 1 → * Team*
- creatore: Concorrente ; *serve per identificare il creatore del team*

Metodi:

+ Team() ; *metodo costruttore, utilizzato per inizializzare un nuovo team appena creato. Imposta in automatico il creatore del team, ossia il concorrente che viene passato come parametro all'interno della funzione.*
+ getNome() ; *metodo utilizzato per visualizzare il valore dell'attributo nome del team invocante la funzione*
+ setNome() ; *metodo utilizzato per impostare un nuovo valore all'attributo nome del team, ma solo tramite il creatore del team stesso*
+ getNome() ; *metodo utilizzato per visualizzare il valore dell'attributo nome del team invocante la funzione*

+ setNome() ; *metodo utilizzato per impostare un nuovo valore all'attributo nome del team, ma solo tramite il creatore del team stesso*
+ getPw() ; *metodo utilizzato per visualizzare il valore dell'attributo pw del team invocante la funzione. Tale metodo può essere utilizzato solo dal creatore del team*
+ setPw() ; *metodo utilizzato per impostare un nuovo valore all'attributo pw del team, ma solo tramite il creatore del team stesso*
+ getCreatore() ; *restituisce il concorrente che ha creato il team*
+ getPunteggio() ; *calcola e restituisce la media dei voti ricevuti dal team*
+ aggiungiMembro() ; *aggiunge un concorrente alla lista dei membri del Team, rispettando il limite massimo di partecipanti*
+ verificaPassword() ; *verifica che la password inserita da un concorrente sia valida per accedere al team richiesto*

Relazioni:

Team * → * Giudice ; *un team può essere giudicato da 0 o più giudici*
Team 1 → * Documento ; *un team può creare 0 o più documenti*
Team * → 1 Hackathon ; *un team appartiene ad un Hackathon*
Team * → 1..* Concorrente ; *un team è composto da almeno un concorrente (fino ad un max specificato dall'Hackathon)*

In particolare, la password non deve mai essere accessibile direttamente dall'esterno, mentre il nome del team deve essere gestito in modo controllato attraverso appositi metodi getter e setter.
I metodi aggiungiMembro() e verificaPassword() verranno poi utilizzati da un metodo contenuto nella classe Concorrente.

La classe Concorrente rappresenta gli utenti che partecipano agli Hackathon creando o unendosi a un team. Questa classe eredita gli attributi e i metodi base dalla superclasse UtentePiattaforma e aggiunge funzionalità specifiche per la gestione dei team.

- La classe Concorrente, presenta i seguenti

Metodi:

- Concorrente() ; *metodo costruttore, utilizzato per inizializzare un nuovo concorrente appena creato*
- creaTeam() ; *permette al concorrente di creare un nuovo team per un determinato Hackathon, impostando un nome e una password. Il metodo verifica che il nome del team sia univoco all'interno dell'Hackathon e aggiunge automaticamente il creatore come primo membro del team*
- partecipaTeam() ; *permette al concorrente di unirsi a un team già esistente inserendo il nome del team e la password corretta. Il metodo verifica che il periodo di iscrizione all'Hackathon non sia terminato e che il team non abbia raggiunto il numero massimo di membri*

Relazioni:

Concorrente 1..* → * Team ; *un concorrente può creare o partecipare a più team nel corso del tempo*

Durante questa fase del progetto non è stata ancora presa in considerazione l'opzione che un concorrente abbandoni un team. Ma per semplicità, qualora esso non voglia più partecipare ad un Hackathon potrà semplicemente non contribuire più alla realizzazione dei documenti (oggetto di giudizio da parte dei giudici) assieme al proprio team.

La classe Hackathon rappresenta l'evento principale della piattaforma. Contiene tutte le informazioni relative all'evento, inclusi i team partecipanti, i giudici assegnati e le date importanti (date relative alle iscrizioni e al periodo di gara)

- La classe Hackathon, presenta i seguenti

Attributi:

- titolo: string
- dataInizio: string
- dataFine: string
- numMaxIscritti: int
- dimMaxTeam: int
- inizioIscrizioni: date
- fineIscrizioni: date
- descrizioneProblema: string
- classifica: string
- indirizzoSede: string

Metodi:

+ Hackathon() ; *metodo costruttore, utilizzato per inizializzare un nuovo Hackathon. Può essere chiamato solo da un organizzatore*

+ getTitolo() ; *metodo utilizzato per visualizzare il valore dell'attributo titolo dell'Hackathon invocante la funzione*

+ setTitolo() ; *metodo utilizzato per impostare un nuovo valore all'attributo titolo dell'Hackathon, ma solo tramite l'organizzatore dell'Hackathon stesso*

...

(omessi, per brevità, i relativi get e set degli attributi restanti)

...

+ getClassifica() ; *restituisce la classifica dell'Hackathon (solo se richiesta da un utente registrato)*

- setClassfica() ; *serve per impostare o aggiornare la classifica dell'Hackathon (invocabile solo dalla funzione calcolaClassifica())*

+ calcolaClassifica() ; *ordina i team in base ai punteggi ricevuti (decrescenti) e genera una classifica (sotto forma di stringa)*
+ esisteTeam() ; *serve a verificare se un determinato team esiste in un Hackathon, restituisce true se il team esiste e false altrimenti*
+ getTeamPerNome() ; *cerca e restituisce un team in base al nome*
+ aggiungiTeam() ; *aggiunge un team alla lista dei team partecipanti*
+ aggiungiGiudice() ; *aggiunge un giudice alla lista dei convocati*
+ isTerminato() ; *serve per verificare se l'Hackathon è terminato (verifica se la data finale dell'Hackathon è stata raggiunta e restituisce true se è terminato, false altrimenti)*
+ iscrizioniTerminate() ; *verifica se le iscrizioni all'Hackathon sono chiuse basandosi sulla data di termine delle iscrizioni e sul numero massimo di partecipanti dell'Hackathon*

Relazioni:

Hackathon * → 1 Organizzatore ; *un Hackathon è creato da un unico organizzatore.*
Hackathon * → * Giudice ; *un Hackathon può essere supervisionato da 0 (nel caso in cui non sia ancora iniziato) o più giudici*
Hackathon 1 → * Team ; *un Hackathon può avere 0 (nel caso in cui sia stato appena creato) o più team partecipanti*

La classe Documento rappresenta i file caricati dai team durante l'Hackathon, come documenti di progetto o presentazioni.

- La classe Documento, presenta i seguenti

Attributi:

- dataAggiornamento: date

- documento: file

Metodi:

+ Documento() ; *metodo costruttore, utilizzato per inizializzare un nuovo documento con un file e impostare automaticamente la data di aggiornamento*

+ getDataAggiornamento() ; restituisce la data dell'ultimo aggiornamento.

- setDataAggiornamento() ; *aggiorna la data associata al documento*

+ getFile() ; *restituisce il file associato al documento*

+ setFile() ; *aggiorna il file associato al documento e imposta automaticamente la data di aggiornamento alla data corrente (tramite setDataAggiornamento)*

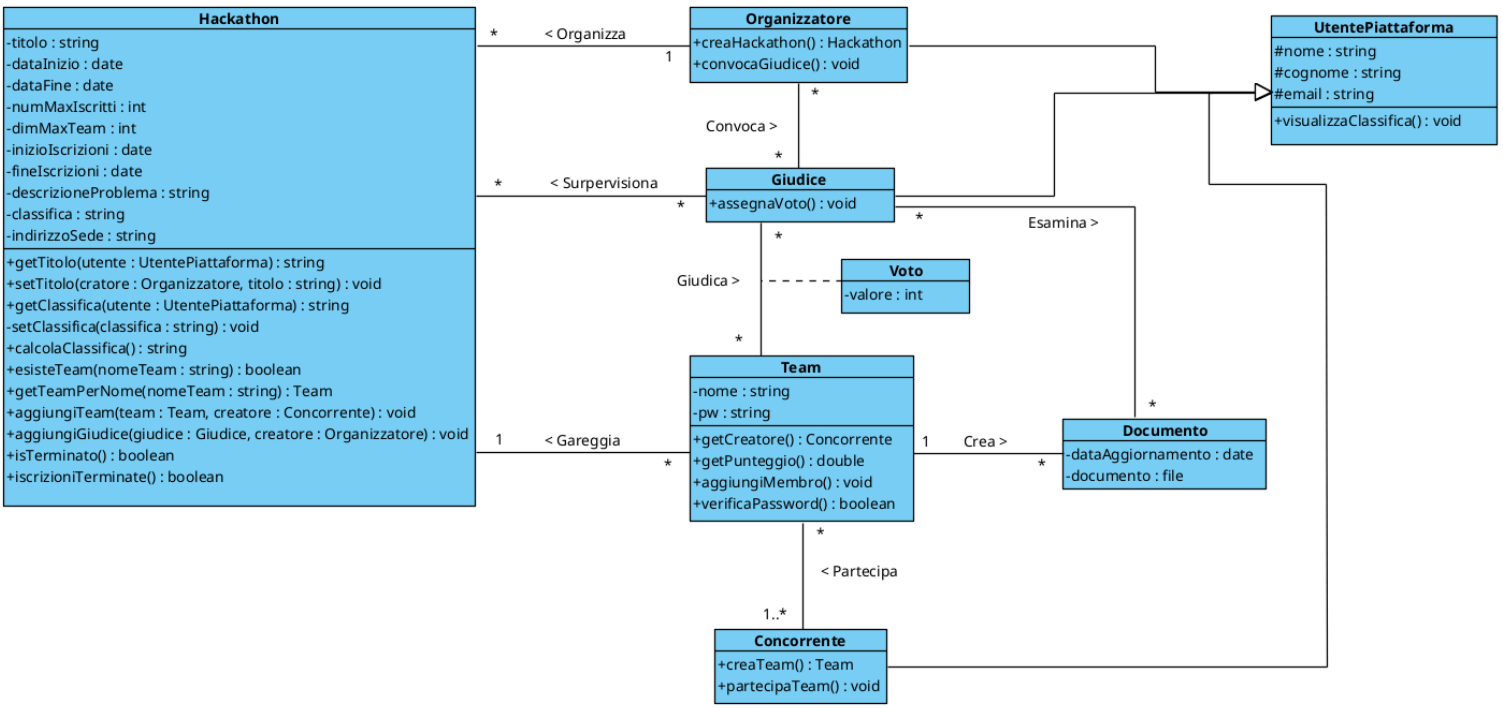
Relazioni:

Documento * → 1 Team ; *un documento viene caricato da un solo team*

Documento * → * Giudice ; *un documento può essere valutato da 0 o più giudici (abilitati a giudicare quel team in quell'Hackathon)*

---

2. Diagramma UML:



NOTA.2:

- ➔ Nel diagramma concettuale UML non sono stati rappresentati i metodi costruttori e tutti i metodi getter e setter, al fine di mantenere il diagramma più pratico e leggibile.
  Tuttavia, in fase di implementazione del codice, tutti i metodi necessari (costruttori, getter e setter ed altri metodi riguardanti la rappresentazione delle relazioni) sono stati regolarmente sviluppati e inseriti.

---

3. Conclusioni:
   La progettazione concettuale del programma (rappresentata attraverso il diagramma delle classi UML), risulta essere coerente con quanto richiesto dalla traccia e supporta correttamente le funzionalità previste.
   Inoltre, la maggior parte delle funzionalità necessarie sono state implementate correttamente, permettendone un utilizzo adeguato.
   Eventuali funzioni aggiuntive potranno essere implementate in una fase successiva del progetto, qualora vengano richieste esplicitamente.

   Per testare il corretto funzionamento delle classi create in Java, è stato fatto utilizzo di un Main di prova.
   Per ulteriori dettagli sul codice sorgente di tali file Java, è possibile consultare la repository GitHub allegata a questo documento, seguendo questi Path:
   src/main/java/model ; *per le classi*
   src/main/java ; *per il Main di prova*

---

4. Link alla directory su GitHub:
   https://github.com/il-Drucoder/Project_OO