DOCUMENTAZIONE HACKATHON

Francesco Piscopo **Raffaele Ruggiero**

CAPITOLI:

1. Analisi della traccia e scelte implementative 2-4
2. Implementazione in Java 5
3. GUI 6-7
4. Database 8

ANALISI DELLA TRACCIA:
Nella traccia possiamo identificare: le classi, le associazioni e gli attributi, le generalizzazioni e le responsabilità

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.

SCELTE IMPLEMENTATIVE DEL DIAGRAMMA:
CLASSI E RESPONSABILITÀ:
Abbiamo scelto di creare una classe **Hackathon** con varie informazioni, ovverosia i team partecipanti, l'organizzatore, varie date, titolo, sede, massimo di iscritti per team, massimo di iscritti, un problema, una classifica e 2 booleani che servono per vedere se le registrazioni sono aperte e se la classifica è stata pubblicata e rispettive funzioni getter e setter.

Oltre alle varie funzioni setter e getter ci sono funzioni più specifiche come 'addTeam' che aggiunge un team alla lista di team partecipanti e 'apriRegistrazioni' che svolge la funzione del suo nominativo.

Abbiamo creato una classe **Team** con un nome, una serie di documenti, una serie di voti e l'hackathon di cui si fa parte. Team è anche una composizione di **Partecipanti**, i quali possono invitare e accettare/rifiutare inviti ad un team, inoltre Partecipante è una delle tre specializzazioni del generico utente.

Team è anche definibile un'aggregazione di **Documento** in quanto un team può aver pubblicato da 0 a N documenti che sono rappresentati da una classe a parte dato che oltre al contenuto del documento in se sono muniti di data e rispettivi commenti.

Utente è una classe munita di email, password e utente univoco. Tutti questi attributi sono protected per poter essere ereditate. L'unico attributo munito di funzione setter è password ma sarebbe semplice implementare anche un setUser e un setEmail, per il primo bisognerebbe però controllare che venga mantenuta la proprietà di unicità dell'attributo.

Il **Giudice** è un'altra classe che estende Utente, le sue responsabilità sono quelle di pubblicare il problema, commentare i progressi dei team e dare voti. Questa classe viene invitata dalla

classe **Organizzatore**, che può anche pubblicare la classifica e aprire le registrazioni.

ASSOCIAZIONI:

Ci sono innanzitutto 4 classi associative: il **Voto**, il **Commento**, l'**Invito** e la **Classifica**.

La classifica è una classe associativa perché è determinata dai team(e i loro voti) e l'hackathon a cui sono iscritti.

Il commento è una classe associativa presente tra il documento e il giudice. Nel nostro scenario inoltre, i giudici possono scrivere un commento a ogni documento e un singolo documento può essere commentato da più giudici.

Il voto è dato dal giudice ed è associato ad un team, ovviamente un giudice da un voto a più team e il team riceve un voto da più giudici.

L'invito è una classe contenente l'invitato, l'hackathon di interesse e il team a cui è stato invitato, a livello implementativo inoltre se il team è vuoto significa che si tratta in realtà di un invito per il ruolo di giudice da parte dell'organizzatore.

Il resto delle associazioni sono abbastanza intuitive (team e hackathon, un team può partecipare a più hackathon e un hackathon ha più team di partecipanti;
un organizzatore invita più giudici e i giudici possono essere invitati da più organizzatori ad hackathon diversi…)

IMPLEMENTAZIONE IN JAVA:

Per l'implementazione in java delle associazioni abbiamo seguito questa logica:

Associazioni 1-1:

Nelle due classi sono presenti un attributo riferito all'altra classe.

In ogni classe che avesse il numero di **partecipazione** pari ad uno abbiamo inserito un costruttore che dasse un valore agli prima menzionati.

Associazioni N-N:

Nelle due classi presenti, hanno un ArrayList di riferimenti all'altra classe.

Nelle classi che necessitano una **partecipazione** minima abbiamo inserito un costruttore che aggiungesse almeno un elemento alla ArrayList.

Composizione:

La classe composta presenta una ArrayList di riferimenti all'altra, mentre l'altra presenta un attributo e un costruttore che si riferiscono alla composta.

GUI:
Per la GUI è stato utilizzato Swing UI Designer, un plugin di IntelliJ Idea.

Ad alto livello la GUI permette ad un utente di eseguire l'accesso
e di registrarsi.

Dopo l'accesso ogni utente può accedere alla sezione in cui può eseguire le sue rispettive funzioni (abbiamo immaginato un utente potesse essere un partecipante di un Hackathon A, un organizzatore di un Hackathon B, e un Giudice di un Hackathon C.

A livello implementativo sono state usate Jlabel, Jcombobox, Jbuttons e Jtables per poter permettere all'utente di interagire con il package Model via una classe chiamata "HackathonController", la quale presenta vari metodi per soddisfare i requisiti degli utenti.

Le classi del package gui sono le seguenti:

1. **LoginForm** che è la pagina contenente il main che tramite altri pulsanti permette di arrivare nel form di registrazione e nella pagina principale.
2. **RegisterForm** che permette la creazione di un utente.
3. **UserInterface** che è la pagina principale contenete vari pulsanti per poter andare in altre pagine, permette di cambiare password, e permette di vedere la classifica degli hackathon tramite una JcomboBox e una table che usa come model la classe **TableClassificaModel**.
4. **Problema** è una GUI accessibile tramite la UserInterface per visualizzare il problema dell'hackathon selezionato nella JcomboBox.
5. **HUDPartecipante** che permette di eseguire le varie funzioni di un partecipante ad un hackathon, come creare un team,

mandare un invito a quest’ultimo, aggiungere un documento e visualizzare i documenti e i commenti annessi loro.

6. **DocumentiCommenti** è la classe a cui si accede via HUDPartecipante.
7. **HUDOrganizzatore** che cambia i valori booleani di uno specifico hackathon e fa invitare i giudici.
8. **HUDGiudice** che permette di pubblicare il problema del singolo hackathon, di assegnare un voto a un team e di accedere ad un’altra GUI per poter commentare i documenti dei team.
9. **ProgressiTeam** che è la gui che viene aperta tramite HUDGiudice.
10. **GUIInviti** che permette ad ogni utente di visualizzare gli inviti ricevuti tramite il modello descritto in **TableInviti** e permette di accettare o rifiutare tali inviti.

DATABASE:
Il database usato nel nostro progetto è PostGreSQL e il programma usato per interfacciarci a postgre è PGAdmin, nel quale tramite 2 query abbiamo definito le varie tabelle con i loro attributi e l'abbiamo popolato tramite vari inserimenti.

In questo progetto viene utilizzato il pattern BCE + DAO nel quale il package controller è l'unico a essere in comunicazione con gli altri, ovvero il DAO, il package Model e la GUI; a livello implementativo abbiamo usato JDBC per gestire la comunicazione con il DBMS.

La connessione al database è implementata tramite il package **db** e la classe **DatabaseConnection** la quale contiene url, username e password come stringhe e usa un pattern singleton.

C'è poi il package **DAO** che è un'interfaccia contenente tutte le firme delle funzioni che gestiscono input e output da database, questo viene poi implementato nel package **DAOImplementation.**

Come regole generali nel package DAOImplementation, la connessione viene aperta e chiusa prima e dopo ogni operazione.

Inoltre ogni funzione che riceve informazioni dal database lo fa usando uno Statement e mettendo le informazioni delle colonne dentro ad una serie di ArrayList inserite nella firma della funzione.

Quando invece si va ad inserire o aggiornare un dato all'interno del database abbiamo usato un PreparedStatement.