Cluster

Realizzare un sistema per gestire l'accesso e l'utilizzo di un cluster per il calcolo. Tutte le classi, eccetto la classe `Esempio`, devono trovarsi nel package `cluster`. Si realizzi il sistema nell'ottica di poter gestire un numero non definito a priori di elementi.

R1. Cluster e Utenti

Il sistema, rappresentato da un oggetto di classe `Cluster`, permette di registrare le informazioni relative a degli utenti, ai loro privilegi all'interno del sistema ed ai processi da essi schedulati. Il costruttore della classe `Cluster` riceve come parametri il numero massimo di processi attivi contemporaneamente per ciascun utente (che siano questi in attesa di essere eseguiti o già in esecuzione) e la lunghezza della coda di esecuzione, che determina il numero massimo di processi in esecuzione nello stesso momento.

Gli utenti del cluster sono rappresentati da oggetti di classe `Utente`, e possono essere distinti in utenti `UtenteAdmin` e `UtenteStandard`. Per registrare un nuovo utente si utilizza il metodo **registraUtente()** della classe `Cluster`, che riceve come parametri le informazioni sull'utente quali email (di cui si deve garantire l'univocità), nome, cognome, password e data di nascita nel formato `AAAAMMGG` (si assuma che il formato del parametro sia corretto). Il metodo si occupa di verificare che la mail sia valida (si assuma per semplicità che una email sia valida qualora contenga il carattere `@`) e che non sia già presente fra quelle degli utenti registrati, che la password sia valida, ovvero che sia lunga non meno di 8 caratteri e che contenga almeno un carattere maiuscolo e un numero, e che l'età del nuovo utente sia di almeno 18 anni (per effettuare tale confronto è sufficiente tenere conto dell'anno di nascita). Nel caso di utente già registrato, il metodo restituisce un oggetto corrispondente all'utente registrato (si consideri un utente come già registrato se esiste un utente con la stessa email); nel caso di nuova registrazione, il valore di ritorno sarà invece rappresentato da un oggetto rappresentante il nuovo utente della tipologia appropriata. In caso di password e/o email non valide o di età inferiore a quella ammessa, il metodo non sortirà alcun effetto ed il valore di ritorno sarà `null`. In caso di `UtenteAdmin` il metodo riceve come parametri aggiuntivi due stringhe identificanti una chiave pubblica e una chiave privata. I vari tipi di oggetto offrono opportuni metodi getter per accedere alle suddette informazioni.

La classe `Cluster` mette a disposizione il metodo **descriviUtente()** che riceve come parametro l'email di un utente e, nel caso in cui questa corrisponda ad un utente registrato nel sistema, restituisce una stringa contenente la descrizione delle sue informazioni. Tale descrizione riporta, separati dalla sequenza di caratteri virgola-spazio `", "`, l'email, il nome, il cognome, la password e la data di nascita. Ai fini della descrizione prodotta da questo metodo, il campo della password è oscurato e sostituito da una serie di caratteri `"*"` di lunghezza pari a quella della password dell'utente che si sta descrivendo. Nel caso di utenti di tipo `UtenteAdmin` la descrizione riporta anche le chiavi (prima la chiave pubblica e poi quella privata). Una variante del metodo riceve come parametri "di autenticazione" una email ed una password e, se queste corrispondono all'email ed alla password di un utente di tipo `UtenteAdmin`, restituisce una descrizione simile a quella precedente ma con il campo password non oscurato. La stessa cosa avviene anche per gli utenti di tipo `UtenteStandard` ma solo nel caso in cui l'email passata come parametro di "autenticazione" sia la stessa dell'utente che si vuole descrivere (ogni utente, salvo l'admin, può descrivere esclusivamente la propria password).

Attraverso il metodo **cercaUtente()** è possibile cercare un certo utente fra quelli registrati; il metodo riceve come parametro l'email dell'utente da cercare e ritorna l'utente desiderato se registrato, `null` altrimenti.

Attraverso il metodo **cercaUtenti()** è possibile ottenere un array contenente gli utenti registrati fino a quel momento nel sistema in ordine di registrazione. Con una seconda versione del metodo, che riceve come parametri aggiuntivi due date (nel formato `AAAAMMGG`), è possibile cercare solo gli utenti la cui data di nascita sia contenuta nel range definito dalle due date. Si supponga che tali date siano passate in ordine crescente e che possano assumere lo stesso valore. Per entrambi i metodi, l'array restituito deve essere dimensionato in base al numero effettivo di utenti trovati.

R2. Processi

Gli utenti registrati nel sistema possono richiedere l'esecuzione di processi da parte del cluster. Ogni processo è caratterizzato da uno specifico stato di esecuzione che può assumere i valori `PENDING`, `ESECUZIONE` o `COMPLETATO`. La classe `Cluster` mette a disposizione il metodo **nuovoProcesso**() che genera un nuovo processo all'interno del cluster, rappresentato dalla classe `Processo`. Il metodo riceve come parametri l'email e la password dell'utente che sta generando il nuovo processo, una stringa riportante la descrizione del processo, un timestamp nel formato `AAAAMMGG HH:MM:SS` che identifica il momento di creazione del processo (si assuma che il formato del parametro sia corretto) e, sulla base di determinate circostanze, schedula il processo. Si consideri inoltre che tale timestamp sarà aggiornato ogni volta che questi processi saranno interessati da un cambio dello stato di esecuzione. In particolare, il metodo verifica che l'email e la password ricevute come parametri corrispondano a quelle di un utente registrato e, in caso positivo, che tale utente non abbia un numero di processi attivi all'interno del cluster superiore a quello ricevuto come parametro nel costruttore della classe `Cluster` (per evitare un'eccessiva monopolizzazione delle risorse). Un processo è considerato attivo se si trova negli stati `PENDING` o `ESECUZIONE`. In caso di esito positivo di tutte le verifiche, il metodo crea un nuovo processo, gli assegna un codice incrementale univoco (a partire da `1`), imposta il suo stato a `PENDING`, lo inserisce all'interno di una coda di

attesa e ne restituisce il riferimento. Questa coda determina l'ordine in cui i processi verranno eseguiti dal cluster e deve essere gestita in modo da eseguire per primi i processi schedulati prima. Il sistema non applica tuttavia le stesse regole per tutti gli utenti. Infatti, nel caso in cui l'email e la password corrispondano a un utente di categoria prioritaria (`UtenteAdmin`) il processo viene inserito in cima alla lista dei processi `PENDING`. La classe `Processo` mette a disposizione opportuni metodi getter per accedere alle informazioni di un processo.

La classe `Cluster` mette a disposizione il metodo **descriviProcesso()** che restituisce la descrizione di un processo all'interno del sistema. Il metodo riceve come parametro il codice di un processo e, se questo corrisponde a un processo del sistema (qualunque sia lo stato), restituisce una stringa contenente la sua descrizione. Tale descrizione riporta, separati dalla sequenza di caratteri virgola-spazio `", "`, il codice del processo, l'email dell'utente che ne ha richiesto l'esecuzione, la descrizione, lo stato più recente ed il relativo timestamp. Nel caso di processo inesistente il metodo non sortisce alcun effetto e il valore di ritorno non è rilevante.

Attraverso il metodo **eseguiProcessi()** della classe `Cluster` è possibile portare uno o più processi dallo stato di `PENDING` allo stato di `ESECUZIONE`. La dimensione della coda di esecuzione (informazione passata come parametro al costruttore della classe `Cluster`) è limitata: il metodo riceve come parametro un timestamp (nello stesso formato dei metodi precedenti) che rappresenta il momento di inizio esecuzione e porta quindi un numero di processi dallo stato di `PENDING` allo stato di `ESECUZIONE` fino allo riempimento della coda. Il metodo restituisce un valore booleano a `true` nel caso in cui almeno un processo venga portato in stato di esecuzione, `false` in caso in cui non ci siano altri processi da eseguire (nessun processo in stato `PENDING`) o la coda dei processi in `ESECUZIONE` sia piena.

Il metodo **completaProcessi()** riceve invece un array di codici dei processi in `ESECUZIONE` ed un timestamp che identifica il momento in cui tali processi vengono completati. Per ognuno dei processi effettivamente esistenti e nello stato di `ESECUZIONE` il metodo aggiorna lo stato a `COMPLETATO` assegnando il suddetto timestamp. Infine, nel caso in cui almeno uno dei processi ricevuti come parametro sia passato dallo stato di `ESECUZIONE` a quello `COMPLETATO`, viene invocato automaticamente il metodo **eseguiProcessi ()** utilizzando il medesimo timestamp. Il metodo restituisce un array di valori booleani della stessa dimesione dell'array ricevuto come parametro. In corrispondenza di processi che sono passati allo stato `COMPLETATO` l'array riporterà il valore `true`, in corrispondenza di processi inesistenti o in stato iniziale diverso da `ESECUZIONE`, il valore false.

Attraverso il metodo **cercaProcesso()** è possibile cercare un processo fra quelli registrati; il metodo riceve come parametro il codice identificativo del processo da cercare e restituisce il processo cercato, se esistente, `null` altrimenti.

Attraverso il metodo **cercaProcessi()** è possibile ottenere un array contenente i processi creati fino a quel momento, in ordine di creazione.

R3. Elenchi Utenti e Processi

La classe `Cluster` mette a disposizione alcuni metodi per descrivere utenti e processi.

Il metodo **elencoUtentiPerEmail()** restituisce una collezione di utenti ordinata per email, in ordine alfabetico inverso.

Il metodo **elencoProcessiPending()** restituisce una collezione dei processi in stato di `PENDlNG` nell'ordine in cui questi saranno eseguiti serviti.

In modo simile, il metodo **elencoProcessiCompletati()** restituisce una collezione dei processi in stato di `COMPLETATO` nell'ordine in cui questi sono stati completati.

Il metodo **elencoProcessiPerTimestamp()** restituisce una collezione dei processi ordinati per timestamp crescenti (dal più vecchio al più recente).

Per tutti i suddetti metodi, nel caso non vi siano utenti/processi da elencare, il valore di ritorno sarà `null`.