Programmazione 24/07/2018

Note sullo svolgimento della prova

- Non è consentito l'uso di vecchi testi d'esame, libri, appunti o dispositivi elettronici personali.
- Utilizzare NetBeans o Notepad++ come editor, e finestre DOS per compilare ed eseguire i programmi.
- La cartella C:\prg contiene il JDK da utilizzare e la documentazione delle Java API. Eventuali file ausiliari preparati dal docente devono essere scaricati da elearn.ing.unipi.it
- I file sorgenti relativi all'esercizio 1 devono essere posti nella cartella C:\prg\myapps\esercizio1 mentre quelli relativi all'esercizio 2 devono essere posti nella cartella C:\prg\myapps\esercizio2.
- Al termine della prova, dopo la sottomissione degli elaborati su elearn.ing.unipi.it, il docente mostrerà una possibile soluzione. Dopo aver visto la soluzione, gli studenti avranno la possibilità di riprendere il proprio elaborato.

Esercizio 1

Una *Barriera* è uno strumento di sincronizzazione che opera su un insieme di *n* thread. In particolare ogni thread dell'insieme, per poter attraversare la barriera, deve attendere che tutti gli altri thread dell'insieme abbiano raggiunto la barriera. Le barriere operano secondo un modello "tutti o nessuno" dal punto di vista della sincronizzazione: se un thread lascia la barriera prematuramente, per esempio perché interrotto, allora tutti gli altri thread in attesa sulla barriera si sbloccheranno in modo anormale attraverso una *BarrieraRottaException* (o una *InterruptedException* se interrotti all'incirca nello stesso istante).
Realizzare una classe *Barriera* che opera secondo le seguenti specifiche:

- *Barriera(int n)*: crea una barriera in cui i thread si bloccano fino a quando il numero di thread in attesa non è pari a *n*.
- *void attendi() throws InterruptedException, BarrieraRottaException*: invocato dai thread quando raggiungono la barriera. Se il thread corrente non è l'ultimo, il thread corrente si blocca fino a quando non si verifica una delle seguenti condizioni:
  - arriva l'ultimo thread;
  - il thread corrente viene interrotto;
  - uno degli altri thread bloccati viene interrotto.

  Se il thread corrente ha il flag interrupted settato quando il metodo viene invocato oppure viene interrotto mentre è in attesa, allora viene lanciata una *InterruptedException*.
  Se un thread viene interrotto mentre è in attesa, allora tutti gli altri thread escono dall'attesa mediante una *BarrieraRottaException*.
- *void reset()*: riporta la barriera nello stato iniziale (la barriera non è più rotta, il numero di thread da attendere torna a *n*). Rilascia eventuali thread in attesa.

Creare una classe di prova *Lavoratore* che opera come segue:

- Dorme per una quantità di tempo casuale compresa tra 0 e 1 secondi;
- Attende che tutti gli altri lavoratori abbiano raggiunto la barriera;
- Stampa un messaggio a video.

Esercizio 2

Realizzare un'applicazione *Depositi Bancari* per consultare un archivio dei depositi dei clienti, in accordo ai casi d'uso di Fig.1 e Fig.2, ai requisiti di Tab.1, e seguendo i medesimi criteri di qualità del progetto del corso.
È possibile consultare esclusivamente le Java API e le slide del corso fornite dal docente in formato pdf.

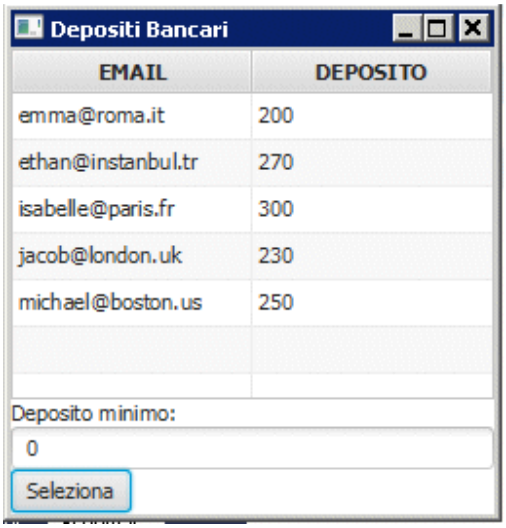

Fig.1 - Primo avvio dell'applicazione:

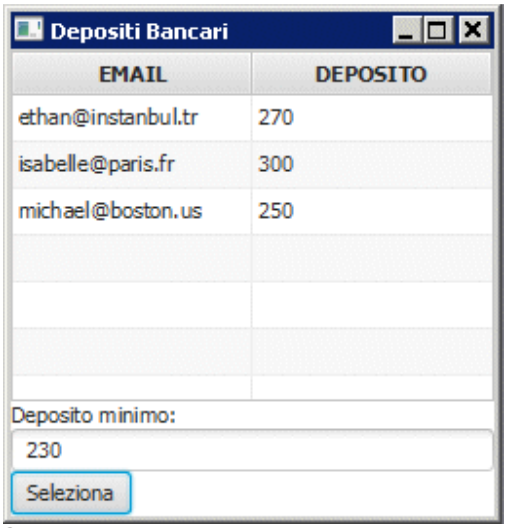

Fig.2 - *Selezione dei clienti con un deposito minimo:*

1. L'Utente avvia l'applicazione
2. FOR EACH utente archiviato
   2.1 Il Sistema visualizza identificativo utente (email) e deposito

1. L'Utente inserisce il Deposito minimo
2. L'Utente preme Seleziona
3. FOR EACH utente archiviato con deposito maggiore di quello minimo
   3.1 Il Sistema visualizza identificativo utente (email) e deposito

Tab.1 - Principali responsabilità e requisiti delle classi da realizzare

Classe	**Principali responsabilità e requisiti**
ConsultazioneDepositiClienti	Costruisce e inizializza il front end dell'applicazione e configura le azioni per ogni evento
TabellaVisualeDepositiClienti	Costruisce e inizializza la tabella dei depositi, la aggiorna a partire da un oggetto List<Cliente>
Cliente	Classe bean
DataBaseDepositiClienti	Costruisce e inizializza la connessione al database e gli statement necessari, riutilizzandoli ad ogni interrogazione e restituendo un oggetto List<Cliente>