

## Note sullo svolgimento della prova

- Non è consentito l'uso di vecchi testi d'esame, libri, o appunti.
- 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

Un prodotto è composto da  $n$  pezzi tutti di tipo diverso. Chiamiamo  $p1$ ,  $p2$ , ...,  $pn$  i diversi tipi di pezzi. Un *Deposito* è in grado di contenere gli  $n$  tipi di pezzi in quantità distinte. In particolare la quantità di pezzi di un determinato tipo va da 0 (quel tipo di pezzo non è presente) a  $m$  (ci sono  $m$  pezzi di quel tipo). In un *Deposito*, per esempio, potrebbero esserci 2 pezzi di tipo  $p1$ , 1 pezzo di tipo  $p2$ ,  $m$  pezzi di tipo  $p3$ , e così via (vedi figura). Dei *Robot* attingono al deposito e realizzano prodotti. Ogni *Robot* cerca di ottenere tutti i pezzi necessari alla realizzazione del prodotto e non appena ha terminato con un prodotto ricomincia. I pezzi vengono immessi nel deposito da un *Fornitore* con disponibilità infinita.

|   |    |    |    |     |    |
|---|----|----|----|-----|----|
| m |    |    | ■  | ... |    |
|   | .  | .  | .  | .   | .  |
| 2 | ■  |    | ■  | ... |    |
| 1 | ■  | ■  |    | ... |    |
|   | p1 | p2 | p3 |     | pn |

Realizzare una class *Deposito* dotata almeno dei seguenti costruttori/metodi:

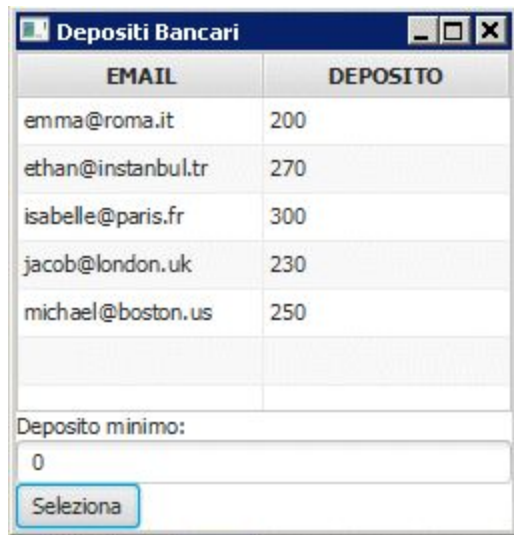
- *Deposito(int n, int m)*: crea un deposito in grado di contenere  $n$  tipi diversi di pezzi; per ogni tipo, la quantità massima di pezzi è pari a  $m$ .
- *deposita()*: chiamato dal Fornitore, inserisce un nuovo pezzo nel deposito; se ci sono dei tipi di pezzi a quantità zero, il *Fornitore* li privilegia rispetto ai tipi in cui c'è almeno un pezzo. Se il deposito è pieno il *Fornitore* si blocca.
- *prendi()*: chiamato dai *Robot*, serve a prendere un pezzo dal deposito limitatamente a quelli che gli mancano per completare il prodotto. Se nessuno dei tipi di pezzi che servono al *Robot* è presente nel *Deposito*, il *Robot* si blocca in attesa di poter proseguire (uno dei pezzi che gli servono è diventato disponibile).

I metodi *deposita()* e *preleva()* possono restituire valori e prendere argomenti a piacimento.

Realizzare anche le classi *Robot* e *Fornitore* secondo la descrizione di cui sopra.

## 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.



| EMAIL             | DEPOSITO |
|-------------------|----------|
| emma@roma.it      | 200      |
| ethan@istanbul.tr | 270      |
| isabelle@paris.fr | 300      |
| jacob@london.uk   | 230      |
| michael@boston.us | 250      |

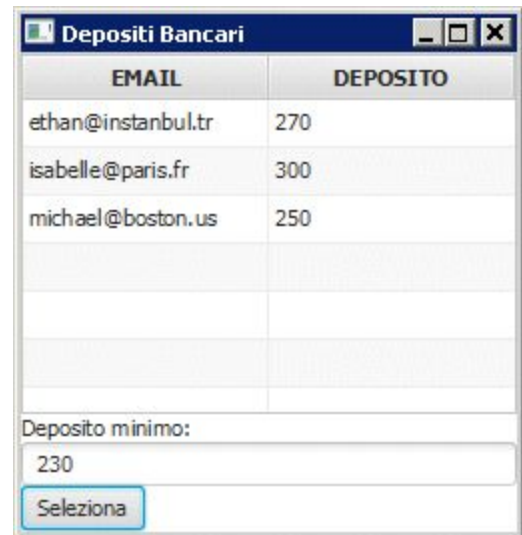
Deposito minimo:

0

Seleziona

Fig.1 - Primo avvio dell'applicazione:

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



| EMAIL             | DEPOSITO |
|-------------------|----------|
| ethan@istanbul.tr | 270      |
| isabelle@paris.fr | 300      |
| michael@boston.us | 250      |

Deposito minimo:

230

Seleziona

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

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> |