Cognome _____ Nome _____
Matricola _____ Postazione PC _____

Corso di Laurea in Ingegneria Gestionale
Esame di Informatica
a.a. 2012-13
14 giugno 2013

Testo

Il database di una banca è costituito da due vettori paralleli. Il primo è denominato “clienti” e contiene oggetti di tipo “Correntista” che rappresentano i correntisti presenti nell’archivio della banca. Il secondo vettore è denominato “conti” e contiene oggetti di tipo “Conto_corrente” che rappresentano le informazioni di un conto corrente di un cliente. Ogni cliente può avere un solo conto corrente all'interno della banca.

Dunque per ogni correntista presente nella posizione i-esima del vettore “clienti” si troveranno le informazioni relative al suo conto corrente nella corrispondente posizione del vettore “conti”. Nel caso che il correntista in posizione i-esima non abbia alcun conto aperto, nella sua posizione nel vettore “conti” sarà presente un riferimento *null*. Entrambi i vettori hanno dimensione pari alla costante “MAX_CLIENTI” (inizializzata a 1024). Se il numero di correntisti contenuti nell’archivio è inferiore a “MAX_CLIENTI”, i primi elementi del vettore conterranno gli oggetti di tipo “Correntista”, mentre gli altri conterranno riferimenti *null*. Tutti gli elementi *null* del vettore “clienti” si devono trovare alla fine del vettore e non possono trovarsi in mezzo agli elementi validi.

Le classe Correntista contiene le informazioni relative ad un correntista ed un metodo per la stampa a video di queste informazioni:

```
public class Correntista {

	public int id;
	public String nome;
	public String cognome;
	public int annoNascita;
	public int meseNascita;
	public int giornoNascita;

	public Correntista(int id, String nome, String cognome, int giornoNascita,
				int meseNascita, int annoNascita) {
		this.id = id;
		this.nome = nome;
		this.cognome = cognome;
		this.giornoNascita = giornoNascita;
		this.meseNascita = meseNascita;
		this.annoNascita = annoNascita;
	}

	public String toString(){
		return "[" + this.id + "] " + this.nome + " " + this.cognome + " ¥t"
		    + this.giornoNascita + "/" + this.meseNascita + "/"
			+ this.annoNascita;
	}
}
```

La classe Conto_corrente contiene le informazioni relative al conto corrente di un correntista.

```
public class Conto_corrente {
	static private int numeroProgressivo = 0;
	private int numeroConto;
	public double saldo;
```

```
        public int annoApertura;
        boolean mutuo;
        public String filiale;

        public Conto_corrente(double saldo, int annoApertura, String filiale,
                              boolean mutuo) {
                numeroConto = numeroProgressivo++;
                this.saldo = saldo;
                this.annoApertura = annoApertura;
                this.filiale = filiale;
                this.mutuo = mutuo;
        }

        public String toString() {
                String tmp = Integer.toString(numeroConto);
                return tmp;
        }
}
```

Si consiglia di procedere nel seguente modo: implementare un metodo e successivamente scrivere la parte del main che utilizza tale metodo, in modo da poterne verificare immediatamente la correttezza.

Le varie operazioni devono essere eseguite sulla porzione significativa dell'archivio, cioè la porzione di "clienti" che non contiene riferimenti "null".

a) **Scrivere il metodo statico:**

```
public static int countCorrentisti(Correntista[] clients)
```

Il metodo deve ritornare il numero effettivo di correntisti presenti nel vettore "clients" (ossia il numero di elementi non nulli assumendo che essi si trovino compattati all'inizio del vettore).

b) **Scrivere il metodo statico:**

```
public static boolean verificaOrdinamentoAnnoNascita(Correntista[] clients)
```

Il metodo deve verificare che nel vettore "clients" gli elementi siano ordinati secondo l'anno di nascita in maniera crescente. Ad esempio, chi è nato nel 1990 viene prima di chi è nato nel 2000. Il metodo deve ritornare true se il vettore è ordinato , false altrimenti.

c) **Scrivere il metodo statico:**

```
public static void ordinaAnnoNascita(Correntista[] clients,
                                      Conto_corrente[] cc_db)
```

Il metodo deve ordinare, nel vettore "clients", gli elementi in maniera crescente secondo l'anno di nascita. Il vettore "cc_db" deve inoltre rimanere consistente. Se il vettore "clients" è già ordìnato, il metodo non deve fare nulla.

d) **Scrivere il metodo statico:**

```
public static boolean eliminaConto(Correntista[] clients,
                                   Conto_corrente[] cc_db,
                                   int idCorrentista)
```

Il metodo deve eliminare dal database specificato dai parametri "clients" e "cc_db" il correntista specificato dal parametro "idCorrentista". Se quel correntista è presente nel database, il metodo deve inoltre restituire "true", altrimenti deve restituire "false".

e) **Scrivere il metodo main che:**

- definisca ed inizializzi i vettori “clienti” e “conti” secondo i valori riportati in tabella e stampi a video l’archivio. La stampa dell’archivio consiste nella stampa delle informazioni di ogni film (usando il metodo “toString” della classe “Correntista”) e, se il correntista possiede un conto, deve essere indicato anche il numero di conto.

Codice	Nome e Cognome	Data di nascita	Saldo	Anno Apertura	Filiale	Mutuo
0	Mario Rossi	15/10/1954	7500	1999	Roma1	No
1	Piero Rossi	15/10/1958				
2	Giovanni Verdi	22/08/1980	500	2007	Roma1	Sì

- Stampi un messaggio che indichi il numero di correntisti usando il metodo *countCorrentisti*.
- Stampi un messaggio che indichi se il database dei correntisti è ordinato, utilizzando il metodo *verificaOrdinamentoAnnoNascista.*
- Ordini l’intero archivio utilizzando il metodo *ordinaAnnoNascita* e stampi a video l’archivio prima e dopo l'ordinamento.
- Elimini il conto corrente relativo al correntista il cui codice è pari a 2 (utilizzando il metodo *eliminaConto*). In caso di esito positivo viene inoltre stampato l’archivio aggiornato.

Soluzione

```
/*
 * Nome
 * Cognome
 * Matricola
 * Postazione
 */

public class Esame {


		static final int MAX_CLIENTI = 1024;

		// scambia l'elemento in posizione i-esima con quello in posizione j-esima
		public static void scambiaConti(Conto_corrente[] v, int i, int j){
			Conto_corrente tmp = v[i];
			v[i] = v[j];
			v[j] = tmp;
		}

		public static void scambiaClienti(Correntista[] v, int i, int j){
			Correntista tmp = v[i];
			v[i] = v[j];
			v[j] = tmp;
		}

		public static int countCorrentisti(Correntista[] clients){
			int counter = 0;
			while (counter < MAX_CLIENTI && clients[counter]!=null){
				counter++;
			}
			return counter;
		}

			/* Soluzione alternativa
			public static int countCorrentisti(Correntista[] clients){
				int counter = 0;
				for (int i=0; i < MAX_CLIENTI; i++){
					if (clients[i] != null){
						counter++;
					}
				}
				return counter;
			}
			*/

		public static boolean verificaOrdinamentoAnnoNascista(Correntista[] clients){
			boolean order = true;
			int n = countCorrentisti(clients);
			for (int i = 0; i < n-1 && order; i++){
				if (clients[i].annoNascita > clients[i+1].annoNascita){
					order = false;
				}
			}
			return order;
		}


		public static void ordinaAnnoNascita(Correntista[] clients, Conto_corrente[] cc_db){
			boolean ordinato = verificaOrdinamentoAnnoNascista(clients);
			if (ordinato == false){
				int n = countCorrentisti(clients);
				for (int i = 0; i < n-1; i++){
					int min = i;
```

```
                                for (int j=i+1; j < n; j++){
                                        if (clients[j].annoNascita< clients[min].annoNascita) {
                                                min = j;
                                        }
                                }
                                scambiaClienti(clients, i, min);
                                scambiaConti(cc_db, i, min);
                        }
                }
        }

                /* Soluzione alternativa
                public static void ordinaAnnoNascita(Correntista[] clients,

Conto_corrente[] cc_db){
                        boolean ordinato = verificaOrdinamentoAnnoNascista(clients);
                        if (ordinato == false){
                                int n = countCorrentisti(clients);
                                for (int i = 0; i < n-1; i++){
                                        for (int j=i+1; j < n; j++){
                                           scambiaClienti(clients, i, j);
                                           scambiaConti(cc_db, i, j);
                                        }
                                }
                        }
                }
                */

        public static boolean eliminaConto(Correntista[] clients,
                                                                        Conto_corrente[] cc,
                                                                        int idCorrentista){
                boolean Trovato = false;
                int n = countCorrentisti(clients);
                int posizione_cliente=-1;
                int i;
                for (i = 0; i < n && !Trovato; i++){
                        if (clients[i].id == idCorrentista){
                                //tengo traccia della posizione
                                // per compattare
                                posizione_cliente=i;
                                clients[i]=null;
                                cc[i] = null;
                                Trovato = true;
                        }
                }

                //compatta l'archivio: metti l'ultimo elemento nella posizione i
                if (Trovato){
                        clients[posizione_cliente]=clients[n-1];
                        cc[posizione_cliente]=cc[n-1];
                        clients[n-1]=null;
                        cc[n-1]=null;
                }

                return Trovato;
        }

        public static void stampaDB(Correntista[] clients,
                                                                Conto_corrente[] cc_db){
                int n = countCorrentisti(clients);
                for (int i=0; i < n; i++){
                        if (cc_db[i] != null){
                                System.out.println(clients[i] + " - cc: " + cc_db[i]);
                        }
                        else {
                                System.out.println(clients[i]);
                        }
                }
        }

        public static void main(String[] args) {
                Correntista[] clienti = new Correntista[MAX_CLIENTI];
                Conto_corrente[] conti = new Conto_corrente[MAX_CLIENTI];

                clienti[0] = new Correntista(0, "Mario", "Rossi", 15, 10, 1954);
                clienti[1] = new Correntista(1, "Piero", "Rossi", 15, 10, 1958);
                clienti[2] = new Correntista(2, "Giovanni", "Verdi", 22, 8, 1980);
```

```
                        conti[0] = new Conto_corrente(7500, 1999, "Roma1", false);
                        conti[2] = new Conto_corrente(500, 2007, "Roma1", true);

                        System.out.println("Il numero dei correntisti è " +
                                                                countCorrentisti(clienti));

                        boolean order = verificaOrdinamentoAnnoNascista(clienti);
                        if (order){
                                        System.out.println("Il database è ordinato!\n");
                        }
                        else {
                                System.out.println("Il database non è ordinato!\n");
                        }

                        stampaDB(clienti, conti);

                        ordinaAnnoNascita(clienti, conti);
                        System.out.println("\nDopo l'ordinamento:");
                        stampaDB(clienti, conti);

                        boolean eliminato = eliminaConto(clienti, conti, 2);
                        if (eliminato) {
                                System.out.println("\nDopo l'eliminazione:");
                                stampaDB(clienti, conti);
                        }
                        else
                                System.out.println("\nNessun elemento eliminato");

        }
}
```