Cognome _____ Nome _____
Matricola _____ Postazione PC _____

Corso di Laurea in Ingegneria Gestionale
Esame di Informatica - a.a. 2014
25 Luglio 2014

Testo

Il database di un bar è costituito da due vettori paralleli. Il primo è denominato “tables” e contiene oggetti di tipo “Tavolo” che rappresentano i tavoli presenti all'interno di un locale. Il secondo vettore è denominato “orders” e contiene oggetti di tipo “Ordine” che rappresentano le informazioni di ogni singolo ordine di un tavolo. Ad ogni tavolo può corrispondere più di un ordine, in quel caso le informazioni del tavolo saranno replicate.
Per ogni tavolo presente nella posizione i-esima del vettore “tables” si troveranno le informazioni relative ad un ordine nella corrispondente posizione del vettore “orders”. Nel caso che il tavolo in posizione i-esima non abbia alcun ordine associato, nella posizione corrispondente nel vettore “orders” sarà presente un riferimento *null*. Entrambi i vettori hanno dimensione pari alla costante “MAX_ELEM” (inizializzata a 1024). Se il numero di tavoli contenuti nell’archivio è inferiore a “MAX_ELEM”, i primi elementi del vettore conterranno gli oggetti di tipo “Tavolo”, mentre gli altri conterranno riferimenti *null*. Tutti gli elementi *null* del vettore “tables” si devono trovare alla fine del vettore e non possono trovarsi in mezzo agli elementi validi.

Le classe Tavolo contiene le informazioni relative ad un tavolo:

```
public class Tavolo {
	private int id;
	public String nomeCliente;
	public String cognomeCliente;
	public String posizione;

	public Tavolo(int myId, String nome, String cognome, String posizione){
		this.id = myId;
		this.nomeCliente = nome;
		this.cognomeCliente = cognome;
		this.posizione = posizione;
	}

	public int getId(){
		return id;
	}

	public String toString(){
		return "Tav.#" + id + " (" + posizione + ") "
			+ nomeCliente + " " + cognomeCliente;
	}
}
```

La classe Ordine contiene le informazioni relative ai singoli ordini di un tavolo.

```
public class Ordine {

	static private int numeroProgressivo = 0;
	private int numeroOrdine;

	public String prodotto;
	public int numero;
	public double prezzo;
	public boolean servito;
	public String orario;

	public Ordine(String product, int quantity, double price, String time, boolean served){
		numeroOrdine = numeroProgressivo++;
		prodotto = product;
		numero = quantity;
		prezzo = price;
		orario =  time;
		servito = served;
	}
```

```
        public String toString(){
                return "[ord." + numeroOrdine + "] "
                        + "num. " + numero + " " + prodotto;
        }
}
```

Si consiglia di procedere implementando un metodo e successivamente la parte del main che utilizza tale metodo.
Le varie operazioni devono essere eseguite sulla porzione significativa dell'archivio, cioè la porzione di "tables" che non contiene riferimenti "null".

A) Scrivere il metodo statico:

```
public static int secondi(String orario)
```

Il metodo deve convertire la Stringa orario passata come parametro nel numero di secondi trascorsi a partire dalla mezzanotte. L'orario può essere interpretabile sia nelle 24h che nelle 12h e dunque può trovarsi nei tre formati *hh:mm*, *hh:mm-AM*, *hh:mm-PM*.

B) Scrivere il metodo statico:

```
public static Ordine[] ordinaArrivo(Tavolo[] tavoli, Ordine[] ordini)
```

Il metodo deve ordinare, nel vettore "ordini", gli elementi in maniera crescente secondo il costo dell'ordine.
Se ci sono ordini nulli questi dovranno trovarsi in fondo al vettore "ordini" e non dovranno essere considerati per l'ordinamento (l'archivio deve essere lasciato in stato consistente). Il metodo deve restituire un vettore ordinato di oggetti di tipo Ordine, allocando memoria per un numero di oggetti classe pari esattamente agli ordini non nulli.

C) Scrivere il metodo statico:

```
public static double contoMaggiorazione(Tavolo[] tavoli,Ordine[] ordini, int id)
```

Il metodo deve restituire il conto totale relativo al tavolo il cui id è specificato come parametro. Nel computo si devono considerare solo gli ordini effettivamente serviti e si deve maggiorare del 20% tutti gli ordini effettuati tra le 22:00 e le 23:59.

D) Scrivere il metodo statico:

```
public static boolean duplica(Tavolo[] tavoli, Ordine[] ordini, int tid1, int tid2)
```

Il metodo deve duplicare gli ordini del tavolo con id tid1 ed assegnarli al tavolo con id tid2, aggiungendoli agli ordini eventualmente già presenti. Se almeno uno dei due tavoli non esiste, il metodo non deve effettuare operazioni e restituisce false, true viceversa.

E) Scrivere il metodo main che:

- definisca ed inizializzi i vettori "tables" e "orders" secondo i valori riportati in tabella. La stampa dell'archivio, ove richiesta, consiste nello stampare le informazioni di ogni tavolo e gli ordini associati (se ve ne sono). Si utilizzino correttamente i relativi metodi toString() implementati nelle due classi. Per ogni ordine si stampi anche l'orario.

Id	Nome e Cognome	Posizione	OrarioOrdine	Prodotto	Quantità	Costo	Servito
0	Mario Rossi	Ristopub	08:56-PM	Acqua	1	2	Sì
1	Pietro Rossi	Pizzeria					
0	Mario Rossi	Ristopub	21:05	Birra	2	3.50	Sì
2	Gio' Verdi	Veranda	10:55-PM	Vino	1	10	Sì
2	Gio' Verdi	Veranda	21:30	Acqua	1	2	Sì

- Avvalendosi del metodo al punto A stampi tutti gli ordini con l'informazione dell'orario convertito.
- Ordini l'intero archivio utilizzando il metodo del punto B e stampi a video gli ordini non nulli in maniera ordinata, includendo l'informazione del costo di ciascun ordine.
- Utilizzando il metodo C, stampi le informazioni relative ai tavoli con id 1 e 2.
- Utilizzando il metodo del punto D, prende gli ordini del tavolo con id 0 e li duplichi nel tavolo con id 2. Ripeta l'operazione tentando di copiare gli ordini del tavolo con id 5 nel tavolo con id 2. Al termine di ogni operazione, se essa è avvenuta con successo si stampi l'archivio aggiornato o viceversa si stampi un messaggio di errore.

Soluzione

```
public class Appello3 {

	static final int MAX_ELEM = 1024;

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

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

	public static int contaTavoli(Tavolo[] tavoli){
		int count = 0;
		while (count < MAX_ELEM && tavoli[count] != null){
			count++;
		}
		return count;
	}

	/**
	 * Restituisce la prima occorrenza del carattere c a partire
	 * da pos
	 */
	public static int indexof(String s, int pos, char c){
		int index = -1;
		boolean trovato = false;
		for (int i = pos; i<s.length() && !trovato; i++){
			if (s.charAt(i) == c){
				index = i;
				trovato = true;
			}
		}
		return index;
	}

	/**
	 * Restituisce la sottostringa di s a partire da pos1 incluso a pos2
	 * escluso
	 */
	public static String substring(String s, int pos1, int pos2){
		String sub = "";
		for (int i = pos1; i<pos2; i++){
			sub += s.charAt(i);
		}
		return sub;
	}

	/*
	 * A. Il metodo deve convertire la Stringa orario nel
	 * numero di secondi trascorsi a partire dalla mezzanotte.
	 * L'orario può essere interpretabile sia nelle 24h che nelle
	 * 12h e dunque può trovarsi nei formati hh:mm, hh:mm-AM,
	 * hh:mm-PM.
	 */
	public static int secondi(String orario){
		int o_s = indexof(orario, 0, ':');
		String ora_s = substring(orario, 0, o_s);
		int ora = Integer.parseInt(ora_s);
```

```
		String min_s = substring(orario, o_s+1, o_s+3);
		int min = Integer.parseInt(min_s);

		int tt_s = indexof(orario, 0, '-');
		if (tt_s != -1){
			String format = substring(orario, tt_s+1, orario.length());
			if (format.equals("PM")){
				ora += 12;
			}
		}

		return (ora * 3600) + (min*60);
	}


	public static int contaOrdini(Ordine[] ordini){
		int count = 0;
		for (int i=0; i<MAX_ELEM; i++)
			if (ordini[i]!= null)
				count++;
		return count;
	}

	/*
	 * B. Il metodo deve ordinare, nel vettore “ordini”, gli elementi in
	 * maniera crescente secondo il costo dell'ordine.
	 * Se ci sono ordini nulli questi dovranno trovarsi in fondo al vettore
	 * “ordini” e non dovranno essere considerati per l'ordinamento (l'archivio deve
	 * essere lasciato in stato consistente). Il metodo deve restituire un vettore
	 * ordinato di oggetti di tipo Ordine, allocando memoria per un numero di oggetti
	 * pari esattamente agli ordini non nulli.
	 */
	public static Ordine[] ordinaArrivo(Tavolo[] tavoli,
			Ordine[] ordini){
		int n = contaTavoli(tavoli);
		for (int i=0; i<n-1; i++){
			for (int j=i+1; j<n; j++){
				if (ordini[i]== null && ordini[j]!= null){
					scambiaTavolo(tavoli, i, j);
					scambiaOrdine(ordini, i, j);
				}
			}
		}

		n = contaOrdini(ordini);
		for (int i = 0; i < n-1; i++){
			int min = i;
			for (int j=i+1; j < n; j++){
				if (costo(ordini[j]) < costo(ordini[min])) {
					min = j;
				}

			}
			scambiaTavolo(tavoli, i, min);
			scambiaOrdine(ordini, i, min);
		}

		Ordine[] ris = new Ordine[n];
		for (int i=0; i<ris.length; i++){
			ris[i] = new Ordine(ordini[i].prodotto, ordini[i].numero,
					ordini[i].prezzo, ordini[i].orario, ordini[i].servito);
		}
		return ris;
	}

	private static double costo(Ordine ordine) {
		double tot = ordine.prezzo * ordine.numero;
		return tot;
	}
```

```java
	/*
	 * C. Il metodo deve restituire il conto totale relativo
	 * al tavolo il cui id è specificato come parametro. Nel computo si
	 * devono considerare solo gli ordini effettivamente serviti e
	 * si deve maggiorare del 20% tutti gli ordini effettuati tra le 22:00
	 * e le 23:59.
	 */
	private static double contoMaggiorazione(Tavolo[] tavoli, Ordine[] ordini,
			int tid) {
		int n = contaTavoli(tavoli);
		double tot = 0;
		final double MAGG = 0.2;
		for (int i=0; i<n; i++){
			if (tavoli[i].getId() == tid){
			 if (ordini[i] != null && ordini[i].servito){
				if (secondi(ordini[i].orario) < secondi("22:00") ||
						secondi(ordini[i].orario) > secondi("23:59"))
						tot += ordini[i].prezzo * ordini[i].numero;
					else {
					  tot += (ordini[i].prezzo + ordini[i].prezzo * MAGG)
						* ordini[i].numero;
					}
				}
			}
		}
		return tot;
	}

	/*
	 * D. Il metodo deve duplicare gli ordini del tavolo con id tid1
	 * ed assegnarli al tavolo con id tid2, aggiungendoli agli ordini eventualmente
	 * già presenti. Se almeno uno dei due tavoli non esiste, il metodo non deve fare
	 * operazioni e restituisce false, true viceversa.
	 */
	public static boolean duplica(Tavolo[] tavoli, Ordine[] ordini, int tid1, int
tid2){
		boolean res = false;
		int n = contaTavoli(tavoli);
		boolean trovato = false;
		int index = -1;
		for (int i=0; i<n && !trovato; i++){
			if (tavoli[i].getId() == tid2){
				trovato = true;
				index = i;
			}
		}

		if (trovato){
			int add = 0;
			for (int i=0; i<n; i++){
				if (tavoli[i].getId() == tid1){
					tavoli[n+add] = new Tavolo(tavoli[index].getId(),
					tavoli[index].nomeCliente,
				  tavoli[index].cognomeCliente, tavoli[index].posizione);
					if (ordini[i]!= null){
						ordini[n+add] = new Ordine(ordini[i].prodotto,
						    ordini[i].numero, ordini[i].prezzo,
						    ordini[i].orario, ordini[i].servito);
					add++;
					}
				}
			}
			if (add > 0)
				res = true;
		}
		return res;
	}
```

```
public static void stampaDB(Tavolo[] tavoli,
              Ordine[] ordini){
       int n = contaTavoli(tavoli);
       for (int i=0; i < n; i++){
              if (ordini[i] != null){
                     System.out.println(tavoli[i] + " - " + ordini[i] + " "
                                   + ordini[i].orario);
              }
              else {
                     System.out.println(tavoli[i]);
              }
       }
}

public static void main(String[] args) {
       Tavolo[] tables = new Tavolo[MAX_ELEM];
       Ordine[] orders = new Ordine[MAX_ELEM];
       tables[0] = new Tavolo(0, "Mario", "Rossi", "Ristopub");
       tables[1] = new Tavolo(1, "Pietro", "Rossi", "Pizzeria");
       tables[2] = new Tavolo(0, "Mario", "Rossi", "Ristopub");
       tables[3] = new Tavolo(2, "Gio'", "Verdi", "Veranda");
       tables[4] = new Tavolo(2, "Gio'", "Verdi", "Veranda");
       orders[0] = new Ordine("Acqua", 1, 2, "08:56-PM", true);
       orders[2] = new Ordine("Birra", 2, 3.50, "21:05", true);
       orders[3] = new Ordine("Vino", 2, 10, "10:55-PM", true);
       orders[4] = new Ordine("Acqua", 1, 2, "21:30", true);

       System.out.println("\nA.");
       for (int i=0; i < MAX_ELEM; i++)
              if (orders[i] != null)
                     System.out.println("La conversione dell'orario per l'ordine "
                            + orders[i]  + " è " + secondi(orders[i].orario));

       System.out.println("\nB.");
       Ordine[] ris = ordinaArrivo(tables,orders);
       for (int i=0; i < ris.length; i++)
          if (ris[i] != null)
              System.out.println(ris[i] + " - costo " + costo(ris[i]) + "€");

       System.out.println("\nC.");
       int tid = 2;
       double totTav0 = contoMaggiorazione(tables, orders, tid);
       System.out.println("Il totale del Tavolo #" + tid + " è " + totTav0 );
       tid = 1;
       totTav0 = contoMaggiorazione(tables, orders, tid);
       System.out.println("Il totale del Tavolo #" + tid + " è " + totTav0 );

       System.out.println("\nD.");
       int tid1 = 0, tid2 = 2;
       boolean res = duplica(tables, orders, tid1, tid2);
       if (res){
          System.out.println("Ordini dal #" + tid1 + " duplicati nel #" + tid2 );
              stampaDB(tables, orders);
       }
       else
          System.out.println("Ordini dal #" +tid1+ " NON duplicati nel #" +tid2 );

       tid1 = 5;
       tid2 = 2;
       res = duplica(tables, orders, tid1, tid2);
       if (res){
          System.out.println("Ordini dal #" + tid1 + " duplicati nel #" + tid2 );
              stampaDB(tables, orders);
       }
       else
           System.out.println("Ordini dal#" +tid1+ " NON duplicati nel #"+tid2 );

}
```