Cognome _____ Nome _____
Matricola _____ Postazione PC _____

Corso di Laurea in Ingegneria Gestionale
Esame di Informatica - a.a. 2014
15 Gennaio 2015

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 elementiSenzaOrdine(Tavolo[] tables, Ordine[] ordini)
```

Il metodo deve effettuare una ricerca tra il 2° ed il 9° elemento del vettore "tables" e restituire il numero di elementi presenti nel suddetto vettore a cui non è associato nessun ordine nel vettore "ordini".

B) Scrivere il metodo statico:

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

Il metodo deve ordinare, nel vettore "tables", gli elementi in maniera crescente usando come criterio la giustapposizione di ora e minuti dell'ordine, interpretata come numero intero. Ad esempio se l'ora è 15:14, il numero da considerare per l'ordinamento è 1514. I tavoli corrispondenti ad ordini nulli si troveranno nella parte alta del vettore. Una stringa può essere convertita in intero tramite la funzione di libreria *int Integer.parseInt(String tmp).*

C) Scrivere il metodo statico:

```
public static void arrotonda(Tavolo[] tables, Ordine[] ordini)
```

Il metodo deve arrotondare il prezzo unitario per ogni ordine associato ai tavoli. L'arrotondamento deve essere effettuato sempre al decimo di centesimo più vicino (es. 2.35 diventa 2.40, 2.34 diventa 2.30; ossia si arrotonda al decimo inferiore se la parte centesimale è minore di 5, a quello superiore se è maggiore o uguale). Per lo svolgimento può essere utile la funzione di libreria *double Math.floor(double d)* che restituisce l'intero inferiore di d espresso come double.

D) Scrivere il metodo statico:

```
public static boolean deleteOra(Tavolo[] tables, Ordine[] ordini, int ora1, int ora2)
```

Il metodo deve eliminare dal db specificato da "tables" e "ordini" tutti gli ordini effettuati tra ora1 ed ora2, estremi inclusi. Se ora1 e ora2 non sono validi, cioè non compresi tra 1 e 24, il metodo non deve effettuare nessuna eliminazione. Se sono entrambi validi ma ora1 è maggiore di ora2 li si dovrà ordinare. Il metodo restituisce true se è stata effettuata almeno una eliminazione, false altrimenti. Anche i rispettivi tavoli devono essere eliminati.

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 orario, prezzo, quantità.

Id	Nome e Cognome	Posizione	OrarioOrdine	Prodotto	Quantità	Prezzo	Servito
0	Mario Rossi	Ristopub	20:56	Acqua	1	2.36	No
1	Pietro Rossi	Pizzeria					
0	Mario Rossi	Ristopub	21:05	Birra	2	3.50	Sì
2	Giovanni Verdi	Veranda	20:55	Vino	2	10.62	Sì

- Avvalendosi del metodo al punto A si stampi il numero di elementi senza un ordine corrispondente.
- Ordini l'intero archivio utilizzando il metodo del punto B e stampi a video l'archivio prima e dopo l'ordinamento.
- Utilizzando il metodo C, si aggiorni l'archivio e si stampi l'archivio aggiornato.
- Utilizzando il metodo del punto D, elimini tavoli e ordini effettuati tra le 21 e le 22. 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 Appello5 {


	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;
	}

	public static String ora(String orario){
		int o_s = indexof(orario, 0, ':');
		String ora_s = substring(orario, 0, o_s);
		return ora_s;
	}

	public static String minuti(String orario){
		int o_s = indexof(orario, 0, ':');
		String min_s = substring(orario, o_s+1, orario.length());
		return min_s;
	}
```

```java
	public static int oraInt(String orario){
		String ora_s = ora(orario);
		int ora = Integer.parseInt(ora_s);
		return ora;
	}


	/*
	 * A. Il metodo deve effettuare una ricerca tra il 2° ed il 9° elemento del
	 * vettore “tables” e restituire il numero di elementi presenti nel suddetto
	 * vettore a cui non è associato nessun Ordine nel vettore “ordini”.
	 */
	public static int elementiSenzaOrdine(Tavolo[] tables, Ordine[] ordini){
		int count=0;
		for (int i=1; i<9; i++){
			if (tables[i]!=null && ordini[i]==null){
				count++;
			}
		}
		return count;
	}

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

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

	/*
	 * B. Il metodo deve ordinare, nel vettore “tables”, gli elementi in maniera
	 * crescente usando come criterio la giustapposizione di ora e minuti dell'ordine,
	 * interpretata come numero intero. Ad esempio se l'ora è 15:14, il
	 * numero da considerare per l'ordinamento è 1514. Una stringa può
	 * essere convertita in intero tramite la funzione di libreria
	 * int Integer.parseInt(String tmp)
	 */
	public static void ordinaTavoli(Tavolo[] tables, Ordine[] ordini){
		int n = contaTavoli(tables);
		for (int i = 0; i < n-1; i++){
			for (int j=i+1; j < n; j++){
				if (ordini[j] == null || ordini[i] != null &&
			trasforma(ora(ordini[j].orario), minuti(ordini[j].orario)) <
			trasforma(ora(ordini[i].orario), minuti(ordini[i].orario)) ) {

					scambiaTavoli(tables, i, j);
					scambiaOrdini(ordini, i, j);
				}
			}
		}
	}


	private static int trasforma(String ora, String min) {
		String vuota = "";
		String tmp = vuota + ora + min;
		return Integer.parseInt(tmp);
	}

	/*
	 * C. Il metodo deve arrotondare il prezzo unitario per ogni ordine associato ai
tavoli.
```

```java
 * L'arrotondamento deve essere effettuato sempre al decimo di centesimo più
 * vicino (es. 2.35 diventa 2.40, 2.34 diventa 2.30; ossia si arrotonda al decimo
 * inferiore se la parte centesimale è minore di 5, a quello superiore se è
 * maggiore o uguale).
 * Si suggerisce di utilizzare per lo svolgimento la funzione di libreria
 * double Math.floor(double d) che restituisce l'intero inferiore di d espresso
 * come double.
 */
public static void arrotonda(Tavolo[] tables, Ordine[] ordini){
	int n = contaTavoli(tables);
	for (int i=0; i<n; i++){
		if (ordini[i] != null){
			double tmp = round(ordini[i].prezzo * 10);
			tmp = tmp/10;
			ordini[i].prezzo = tmp;
		}
	}
}

private static double round(double costo) {
	double floor = Math.floor(costo);
	double decimal = costo - floor;
	double result;
	if (decimal < 0.5)
		result = floor;
	else
		result = floor + 1 ;
	return result;
}

/*
 * D. Il metodo deve eliminare dal db specificato da “tables” e “ordini” tutti
 * gli ordini di Tavoli effettuati tra ora1 ed ora2, estremi inclusi.
 * Se ora1 e ora2 non sono parametri validi, cioè non compresi tra 1 e 24, il
 * metodo  non deve effettuare
 * eliminazioni. Se sono entrambi validi ma ora1 è maggiore di ora2 li si dovrà
 * ordinare.
 * Il metodo restituisce true se è stata effettuata almeno una eliminazione, false
 * altrimenti.
 * Anche i rispettivi Tavoli devono essere eliminati (lasciando l'archivio
 * in uno stato consistente).
 */
public static boolean deleteOra(Tavolo[] tables, Ordine[] ordini,
		int ora1, int ora2){
	boolean eliminati = false;
	int tmp;
	if (ora1 >= 1 && ora1 <= 24){
		if (ora2 >= 1 && ora2 <= 24){
			if (ora1 > ora2){
				tmp = ora2;
				ora2 = ora1;
				ora1 = tmp;
			}
			int n = contaTavoli(tables);
			for (int i=0; i<n; i++){
				if (ordini[i] != null &&
					oraInt(ordini[i].orario) >= ora1 &&
					oraInt(ordini[i].orario) <= ora2){
					tables[i] = null;
					ordini[i] = null;
					eliminati = true;
				}
			}

			if (eliminati){
// poichè si sono eliminati anche dei Tavoli, si deve fare una compattazione
				for (int i=0; i<n-1; i++){
					for (int j=i+1; j<n; j++){
						if (tables[i]== null && tables[j]!= null){
```

```
                                                scambiaTavoli(tables, i, j);
                                                scambiaOrdini(ordini, i, j);
                                          }
                                    }
                              }
                        }
                  }
            }
            return eliminati;
      }

      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 + " "
                                    + ordini[i].prezzo + "€");
                  }
                  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, "Giovanni", "Verdi", "Veranda");

            orders[0] = new Ordine("Acqua", 1, 2.36, "20:56", false);
            orders[2] = new Ordine("Birra", 2, 3.5, "21:05", true);
            orders[3] = new Ordine("Vino", 2, 10.62, "20:55", true);

            System.out.println("\nA.");
            System.out.println("Il numero di elementi (compresi tra il 3° ed il 10°) "
+
                        "nel vettore \"tavoli\" senza Ordine" +
                        " è " + elementiSenzaOrdine(tables, orders));

            System.out.println("\nB.");
            ordinaTavoli(tables, orders);
            stampaDB(tables, orders);

            System.out.println("\nC.");
            arrotonda(tables, orders);
            stampaDB(tables, orders);

            System.out.println("\nD.");
            boolean eliminati = deleteOra(tables, orders, 21, 22);
            if (!eliminati)
                  System.out.println("\nErrore nell'operazione precente");
            else
                  stampaDB(tables, orders);
      }
}
```