

Programmazione II, 28 giugno 2011 – Laurea in INFORMATICA**Esercizio 1**[16 punti]

Una coda è una sequenza (a_1, a_2, \dots, a_n) di zero o più elementi dello stesso tipo, in cui è possibile aggiungere elementi ad un estremo della sequenza e toglierli dall'estremo opposto. L'inserimento in una coda segue la politica FIFO (First In First Out).

Realizzare una classe che implementa una coda per oggetti generici, cioè della classe `Object`. La classe dovrà contenere

- un array di oggetti di capacità prefissata che memorizza nelle prime n posizioni gli n elementi contenuti nella coda; la dimensione dell'array può essere richiesta come dato da leggere in input per la costruzione di una particolare istanza, oppure fissata come costante di classe;
- una variabile `head` che contiene l'indice dell'elemento in testa alla coda, e una variabile `tail` che contiene l'indice dell'elemento in fondo alla coda.

La classe dovrà implementare la seguente interfaccia:

```
public interface IntCoda {  
  
    //restituisce il numero di elementi nella coda  
    public int size ();  
  
    //controlla se la coda e' vuota  
    public boolean isEmpty();  
  
    // restituisce l'elemento iniziale della coda senza eliminarlo dalla coda  
    public Object front () throws EmptyQueueException;  
  
    //inserisce un elemento alla fine della coda  
    public void enqueue (Object element);  
  
    // rimuove l'elemento all'inizio della coda  
    public Object dequeue() throws EmptyQueueException;  
  
}
```

L'eccezione lanciata rispettivamente dai metodi `front` e `dequeue` è implementata dalla classe

```
public class EmptyQueueException extends RuntimeException {  
  
    public EmptyQueueException(String err) {  
super(err);  
    }  
}
```

Per l'implementazione dei metodi definiti dall'interfaccia `IntCoda`, si assuma che l'array sia circolare, cioè se N è la dimensione dell'array, l'elemento in posizione 0 è successivo a quello in posizione $N - 1$ (ultimo elemento dell'array). Pertanto,

- gli incrementi della variabile `head` (in `dequeue`) e `tail` (in `enqueue`) devono essere operazioni del tipo $(x + 1) \bmod N$;
- il numero effettivo degli elementi contenuti nella coda, `size()`, è dato dalla formula $(N - \text{head} + \text{tail}) \bmod N$.

Per l'operazione `enqueue`, si assuma che la coda è piena quando tutte le posizioni sono state occupate tranne una, cioè quando `size() = N - 1`. In questo caso il metodo lancia un'eccezione di tipo `FullQueueException` definita in modo analogo alla eccezione definita sopra.

Infine, la condizione `head = tail` indica che la coda è vuota.

Esercizio 2[16 punti]

Un Centro Assistenza è una struttura che dispone di un meccanico e un autolavaggio mediante i quali svolge funzioni di riparazione e lavaggio macchine.

Date le seguenti interfacce:

```
public interface Lavabile {

    public boolean isDaLavare (); //stabilisce se l'oggetto e' da lavare
    public void lavaggio(); //stampa un messaggio che annuncia che l'oggetto e' stato lavato
    public void setDaLavare(boolean b); //aggiorna lo stato dell'oggetto a 'non-da-lavare'
}

public interface Riparabile {

    public boolean isFunzionante (); //stabilisce se l'oggetto funziona
    public void riparazione(); //stampa un messaggio che annuncia che l'oggetto e' stato riparato
    public void setFunzionante(boolean b); //aggiorna lo stato dell'oggetto a 'funzionante'
}
```

si definisca una classe `CentroAssistenza` che contiene due variabili di istanza private di tipo `Meccanico` e `AutoLavaggio`, rispettivamente; un costruttore che le inizializza, e due metodi con i seguenti prototipi

- `public void riparazione(Riparabile r)` e
- `public void lavaggio(Lavabile l)`

che invocano le rispettive funzioni delle variabili d'istanza.

Si definiscano anche le classi `Meccanico` e `AutoLavaggio`. La prima consiste di un solo metodo, `public void riparazione(Riparabile r)`, che controlla se l'oggetto passato come parametro è da riparare e in questo caso lo ripara e ne aggiorna lo stato. Analogamente, la seconda classe consiste di un solo metodo, `public void lavaggio(Lavabile r)`, che controlla se l'oggetto passato come parametro è da lavare e in questo caso lo lava e ne aggiorna lo stato.

Si scriva infine una classe `Automobile` che implementa le interfacce `Lavabile` e `Riparabile`. La classe contiene:

- una variabile privata, `nome` di tipo `String`, che contiene il nome dell'istanza,
- due variabili private, `isDaLavare` e `isFunzionante` di tipo `boolean`, il cui valore è `true` se e solo se l'oggetto è rispettivamente da lavare o funzionante,
- un costruttore `Automobile(String n)` che inizializza `nome` col valore passato come parametro.