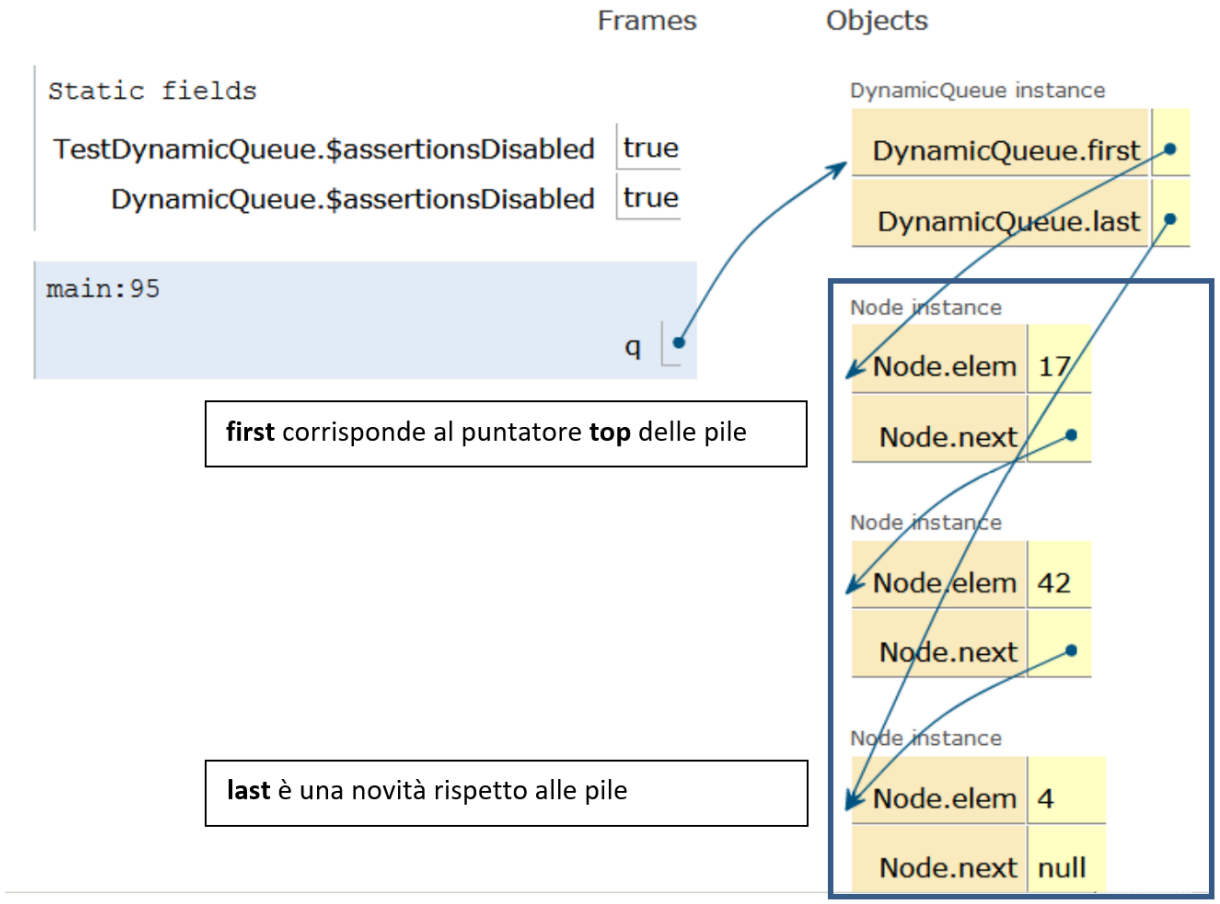
Code dinamiche

Una coda è una struttura dati in cui gli elementi vengono inseriti/rimossi secondo la politica ***FIFO (First-In-First-Out):*** il primo elemento inserito è il primo a essere rimosso. Una coda viene usata per eseguire dei compiti nello stesso ordine con cui si presentano.

Vi chiediamo di definire una implementazione ***DynamicQueue*** delle code dinamiche usando la classe di nodi vista in precedenza, e adattando l'implementazione delle pile dinamiche vista nella lezione precedente. Una coda dinamica viene definita come una lista di nodi (anche vuota) in cui ogni nome punta al ***successivo*** (come nella pila) con due attributi privati: un puntatore ***first*** al primo elemento della coda (il primo ad essere eliminato) e un puntatore ***last*** all'ultimo elemento della coda, l'ultimo arrivato, dietro al quale aggiungeremo il prossimo elemento.

Potete immaginare una coda dinamica come una pila dinamica dove top viene chiamato first e dove abbiamo un nuovo puntatore, last, che punta alla fine della coda.

Vediamo il disegno una coda {17,42,4}, con il nodo "first" che contiene 17 in alto e con il nodo "last" che contiene 4 in basso. Una oggetto di tipo coda punta a una coppia di indirizzi first, last, muovendoci a partire da first nell'esempio troviamo 17 e l'indirizzo del nodo che contiene 42, in quest'ultimo l'indirizzo del nodo che contiene 4 e null (la fine della lista). Possiamo arrivare a 4 in un passo solo se seguiamo il puntatore last.



La coda q={17,42,4}: "first" punta a 17 e "last" punta a 4

Tutti i metodi di **DynamicQueue** sono pubblici e dinamici. Definite ***(i)*** un costruttore per la coda vuota, ***(ii)*** un metodo di scrittura, ***(iii)*** un metodo ***void enqueue(int x)*** per aggiungere un elemento dietro l'ultimo, ***(iv)*** un metodo ***int dequeue()*** per togliere il primo elemento della coda, ***(v)*** un metodo ***int size()*** per contare gli elementi della coda, ***(vi)*** un metodo ***int front()*** per leggere il primo elemento della coda senza toglierlo ***(vii)*** un metodo ***boolean empty()*** per verificare se la coda è vuota.

Suggerimento. A differenza delle pile e code definite tramite array, i cicli per scorrere la struttura non usano indici interi, ma i puntatori/indirizzi, che vengono spostati all'elemento successivo (si veda il codice). **Facoltativo.** Definite un metodo pubblico ***boolean contains(int x)*** per verificare se la coda contiene un dato elemento x.

Tutti i metodi devono preservare il seguente ***invariante della classe***: ogni nodo tranne l'ultimo punta al ***successivo***, e first e last puntano al primo e all'ultimo elemento della coda. Inoltre first e last sono uguali a ***null*** se la coda è vuota.

Esecuzione q.enqueue(4) con q={17,42}

Vediamo come cambia una coda q={17,42} se aggiungiamo 4 in fondo. Il puntatore first non cambia, il puntatore last che puntava al nodo che contiene 42 ora punta al nodo che contiene 4. Anche il nodo contenente 42, che puntava a null ed era alla fine della coda, ora punta al nodo che contiene 4.

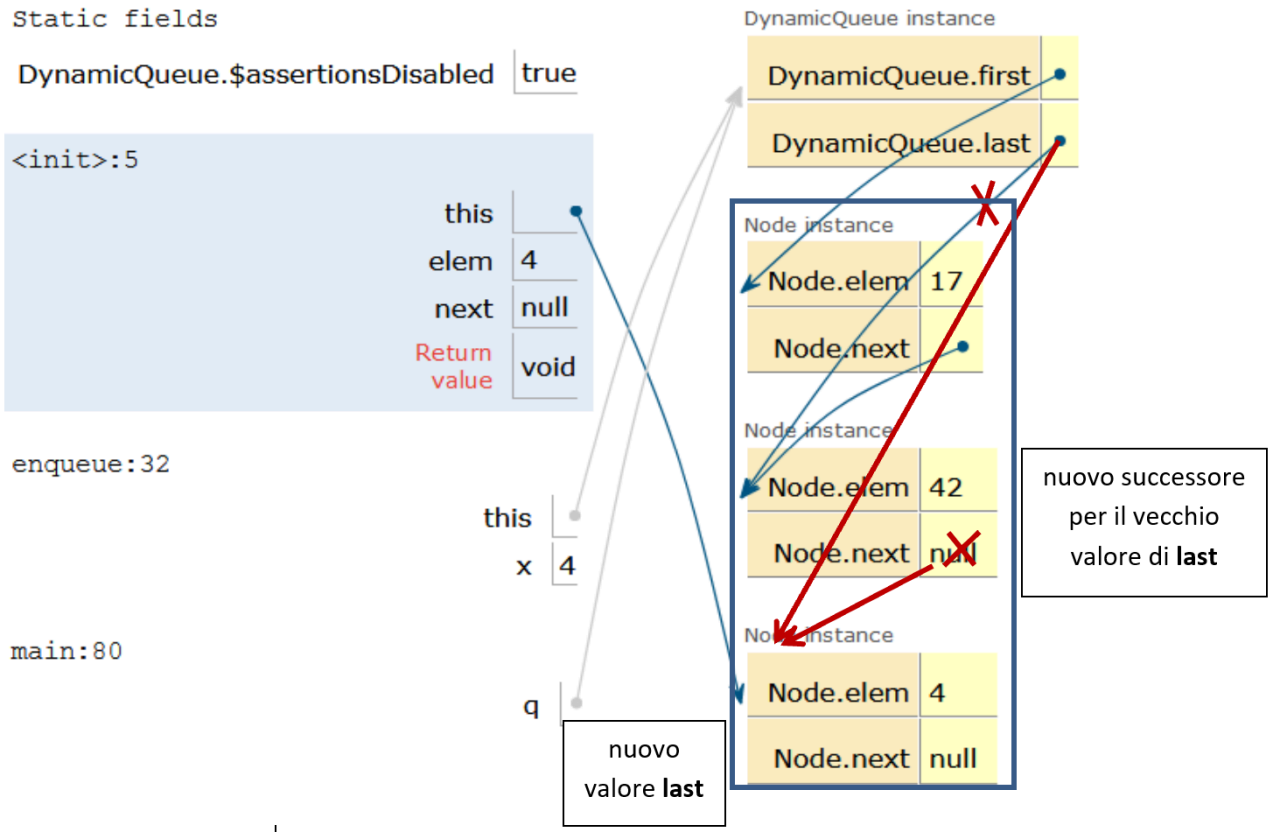
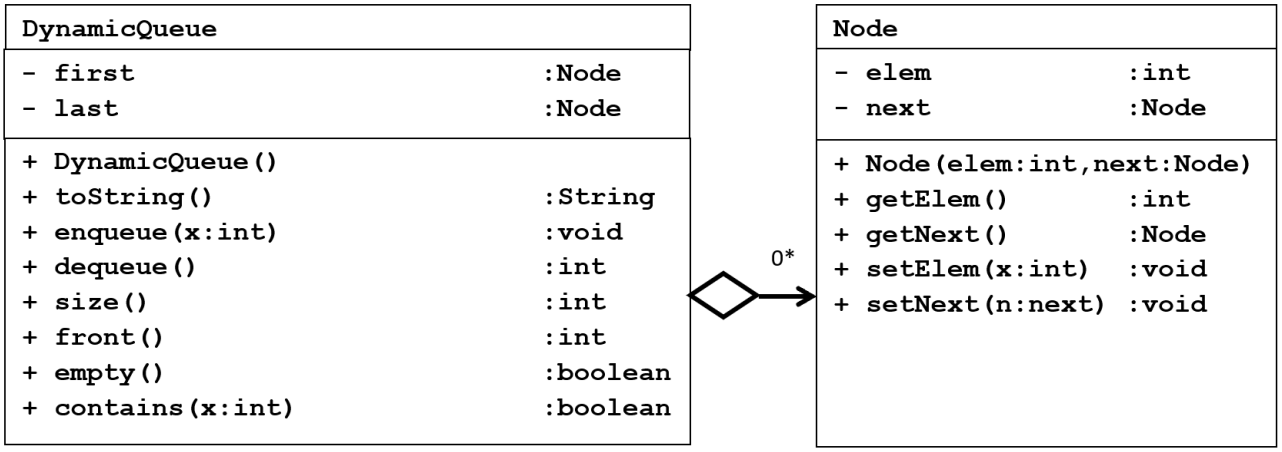


Diagramma UML per le code dinamiche

Nel diagramma indichiamo che una coda dinamica è definita ***aggregando*** 0 o più elementi della classe Node.



Usate la classe TestDynamicQueue inclusa qui sotto come test per la classe DynamicQueue.

```
//Node.java
//Riutilizzate la classe Node definita nella Lezione 08

//TestDynamicQueue.java
//Usate questa classe come test per DynamicQueue
public class TestDynamicQueue{
 public static void main(String[] args){
   DynamicQueue q = new DynamicQueue();
   System.out.println( "q = {17,42,4} " );
   q.enqueue(17); q.enqueue(42); q.enqueue(4);
   System.out.print(q);
   System.out.println( "q.empty() = " + q.empty());
   /** Aggiungete queste righe se avete realizzato "contains"
 System.out.println( "q.contains(4)  = " + q.contains(4)); //true
 System.out.println( "q.contains(40) = " + q.contains(40));//false
*/
   System.out.println("q.size() = " + q.size());  // stampa 3
   System.out.println("q.front()= " + q.front()); // stampa 17
   System.out.println(q.dequeue()); //toglie e stampa 17
   System.out.println(q.dequeue()); //toglie e stampa 42
   System.out.println(q.dequeue()); //toglie e stampa 4: coda
vuota
   // gli elementi vengono stampati nello stesso ordine in cui
   // sono stati inseriti, dal momento che la coda e' una
   // struttura FIFO (First-In-First-Out)
```

```
    System.out.println( "q.empty() = " + q.empty());
    /** Questo comando deve far scattare un "assert":
    q.front();*/
  }
}
```