Programmazione II

Esercitazione 02: Code Dinamiche

Alessandro Mazzei

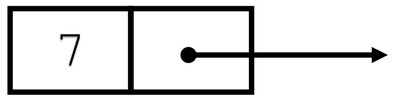
Slides Credit: Daniele Radicioni

breve sintesi dei temi oggetto dell’esercitazione

- Node
- DynamicStack
- DynamicQueue

- Node
- DynamicStack
- DynamicQueue

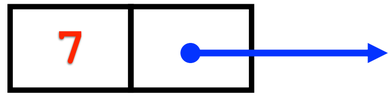
la classe Node



- Nodo: coppia costituita da un dato (un intero) e da un puntatore ad altro Nodo.
  - questa definizione può essere implementata con due campi: un valore, e un campo next (che contiene il riferimento a un altro nodo, o a `null` nell'ultimo elemento).

```
public class Node {
   private int elem;
   private Node next;
   ...
}
```

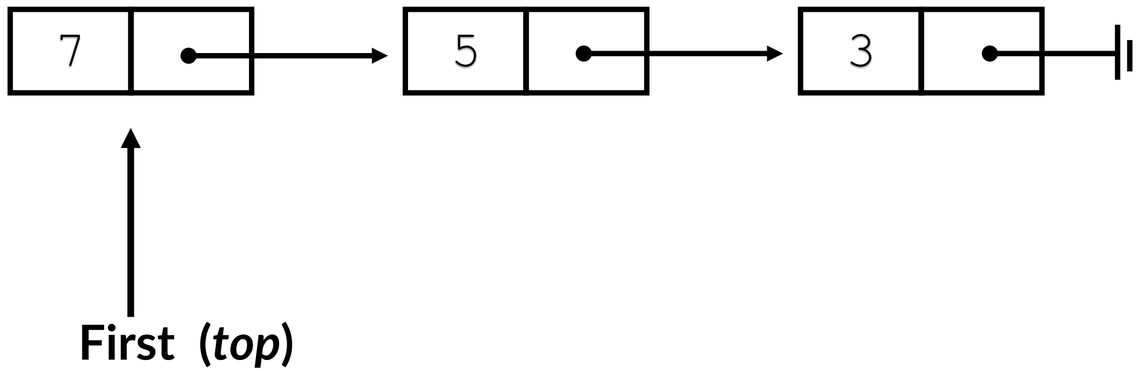
la classe Node




```
//Node.java
public class Node {
  private int elem;
  private Node next;

  public Node(int elem, Node next){this.elem=elem; this.next=next;}

  public int getElem(){return elem;}
  public Node getNext(){return next;}
  public void setElem(int elem){this.elem=elem;}
  public void setNext(Node next){this.next=next;}
}
```

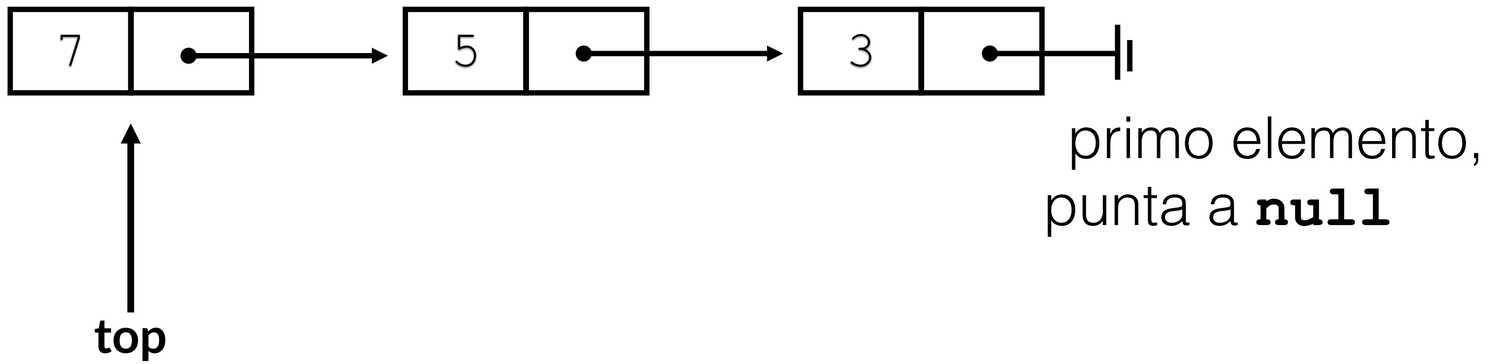
- Node
- DynamicStack
- DynamicQueue

la classe DynamicStack



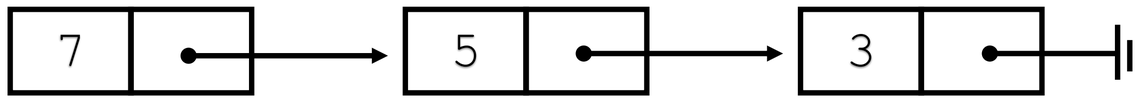
- Tutte le operazioni su Stack siano condotte utilizzando una politica LIFO (Last-In-First-Out), in cui **sia inserimenti sia rimozioni sono condotti tramite il first (*top)***.

la classe DynamicStack



- numero arbitrario di elementi
  - DynamicStack è una lista di nodi, ciascuno dei quali contiene un valore e un riferimento all'elemento precedente (salvo il primo elemento, che punta a **`null`**).
  - DynamicStack ha un **`top`** (non nullo se stack non è vuoto) che punta all'ultimo elemento della pila.

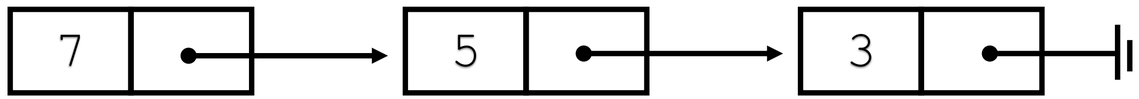
la classe DynamicStack



- i metodi tipici per maneggiare uno stack sono
  - **`void push(int x)`**, aggiunge un elemento (*si agisce solo e sempre in cima allo stack, mai sugli elementi interni*);
  - **`int pop()`**, elimina il nodo in cima, restituendone il contenuto;
  - **`int top()`**, restituisce il contenuto del nodo in cima **senza alterare** stack;
  - **`boolean empty()`**, verifica se la pila è vuota.

```
public class DynamicStack{
  private Node top; // ultimo nodo aggiunto; "null" se pila vuota
  ...
}
```

la classe DynamicStack

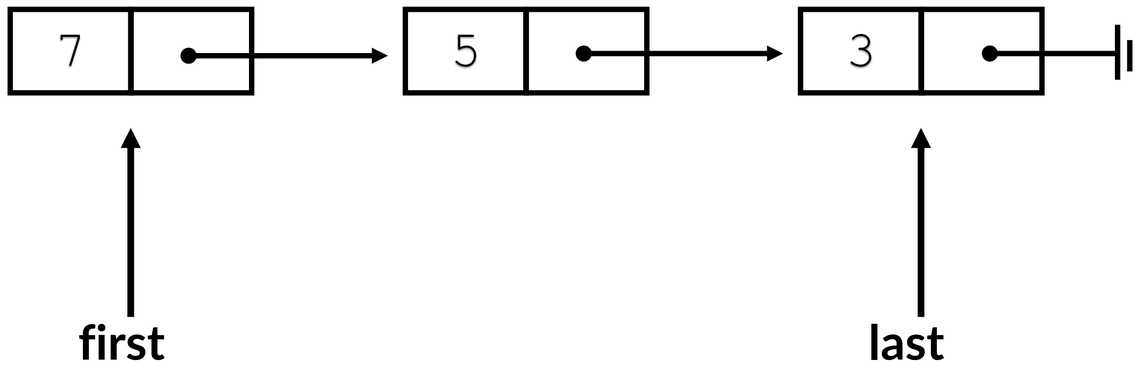



```
//aggiungo un nodo in cima alla pila con un nuovo elemento x
public void push(int x) {top = new Node(x,top);}

//tolgo il nodo in cima alla pila e restituisco il suo contenuto
public int pop(){
 assert !empty();
 int x = top.getElem();
 top = top.getNext(); //elimino l'ultimo nodo con contenuto x
 return x;
}
```

- Node
- DynamicStack
- DynamicQueue

- *Politica*
- *Inserimento*
  - *Primo elemento*
- *Estrazione*
  - *Ultimo elemento*
- *Scorrimento*

la classe DynamicQueue



- politica FIFO (First-In-First-Out):
  - inserimento in coda e rimozione in testa;
  - ci sono ora due puntatori (non più solo il first): **first**, elemento da estrarre, e **last**, elemento da accodare.

esempio di politica FIFO: la coda alla Posta...

esempio: coda alla Posta...

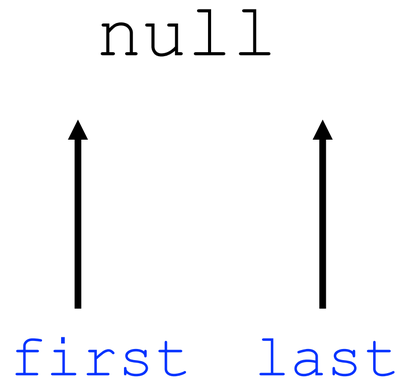
```
public class ClientePosta {
   private String nomeCognome;
   private ClientePosta next;
   ...
}
```

esempio: coda alla Posta...

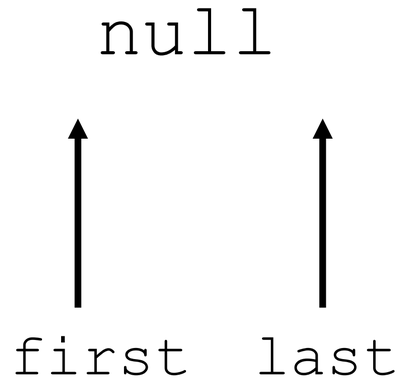
```
public class CodaAllaPosta {
   private ClientePosta first;
   private ClientePosta last;
   ...
   public void inserisci(ClientePosta cliente) {...}
   public ClientePosta estrai() {...}
   ...
}
```

creazione e aggiunta primo elemento

- inizialmente, quando la coda è vuota, sia first sia last puntano a null



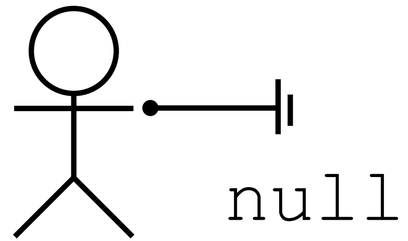
creazione e aggiunta primo elemento



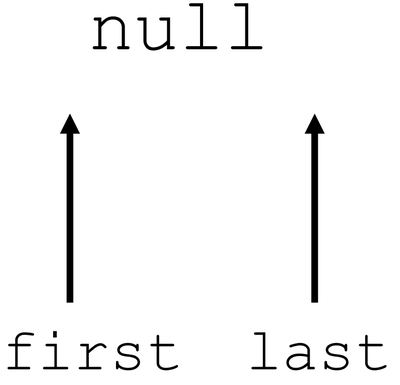
- inizialmente, quando la coda è vuota sia first sia last puntano a null

- quindi viene istanziato il primo cliente

```
ClientePosta primoDellaFila
        = new ClientePosta("IvoGialli", null);
```



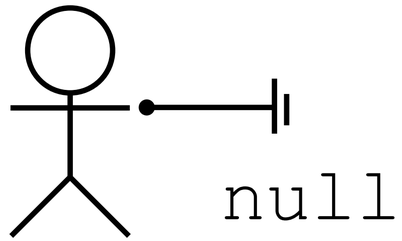
creazione e aggiunta primo elemento



- inizialmente, quando la coda vuota sia first sia last puntano a null

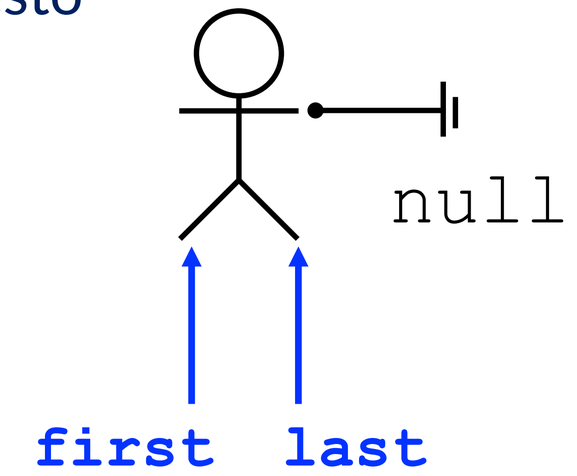
- quindi viene istanziato il primo cliente

```
ClientePosta primoDellaFila
        = new ClientePosta("IvoGialli", null);
```



- il primo cliente viene inserito in coda: sia first sia last puntano a questo primo elemento Nodo/ClientePosta

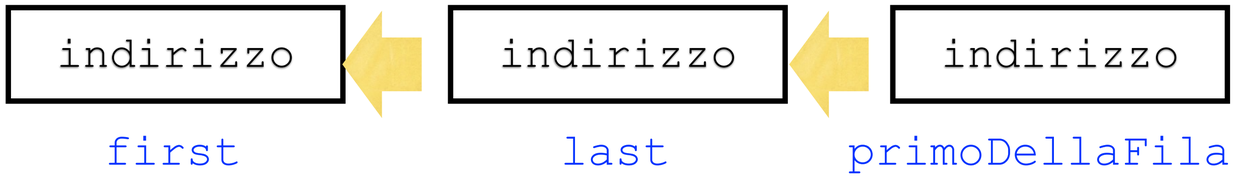
```
first = last = primoDellaFila;
```

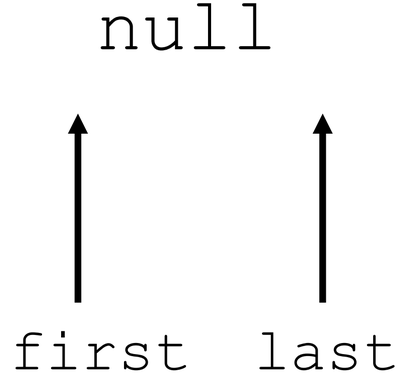


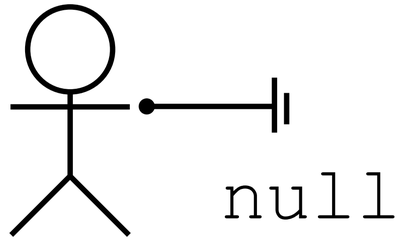
creazione e aggiunta primo elemento

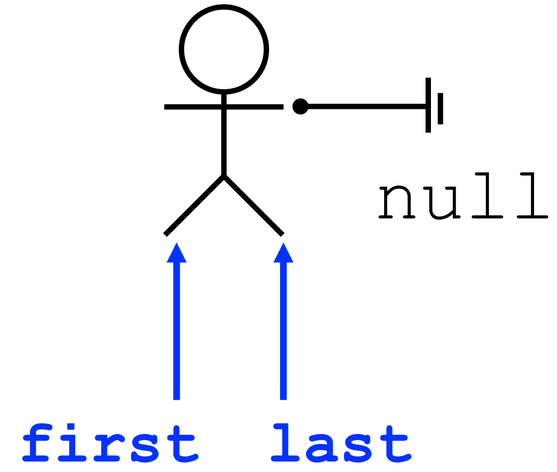
```
first = last = primoDellaFila;
```

... **attenzione** a questo assegnamento!!!
operatore = **associa da dx a sx**: questo significa che il valore di `primoDellaFila` viene copiato in `last`, e il valore di `last` viene quindi copiato in `first`.
una rappresentazione grafica potrebbe essere quella qui sotto:



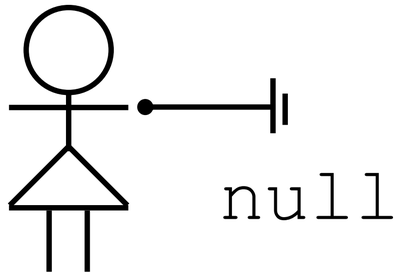




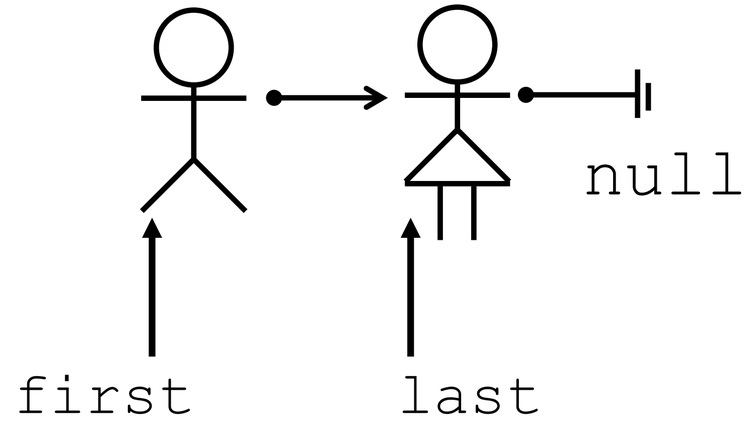


altri inserimenti

A.



B.



- viene (A.) istanziato e (B.) accodato **un altro cliente**
- nel primo caso eseguiamo

```
newCliente = new ClientePosta("MarioRossi", null);
```

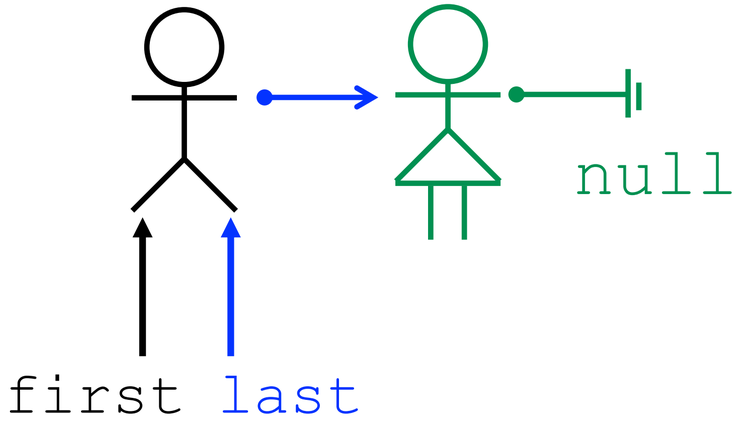
- la seconda operazione potrebbe essere condotta con il metodo

```
inserisci(newCliente) {...}
```

inserimento

- consideriamo l'inserimento: 2 operazioni importanti
  - `last.next = newCliente;`

B.



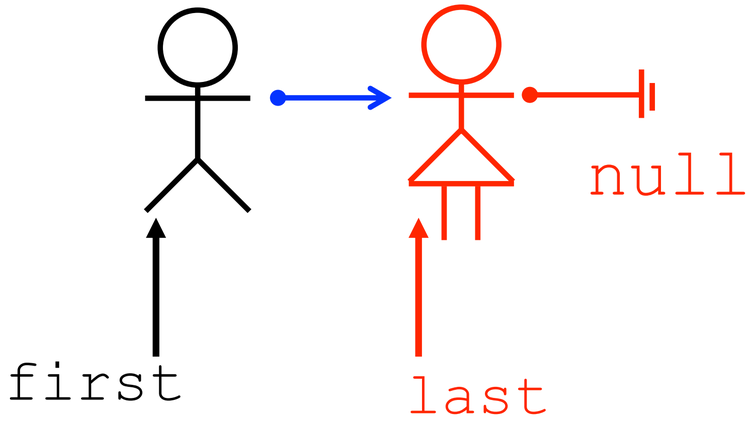
inserimento

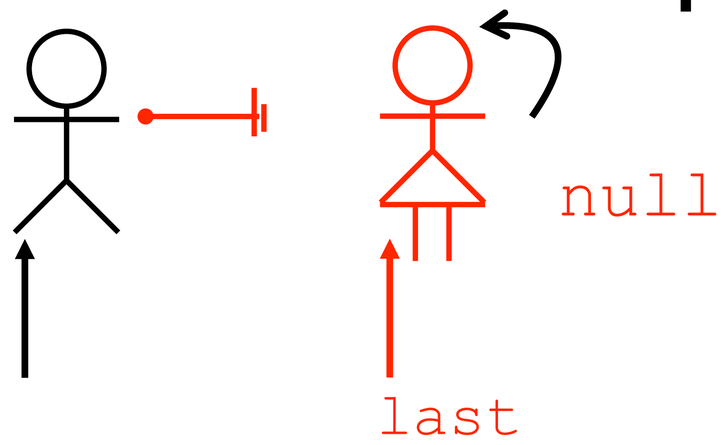
- consideriamo l'inserimento: 2 operazioni importanti
  - `last.next = newCliente`
  - `last = newCliente`

- NB: le ***stesse operazioni in ordine invertito NON FUNZIONANO***

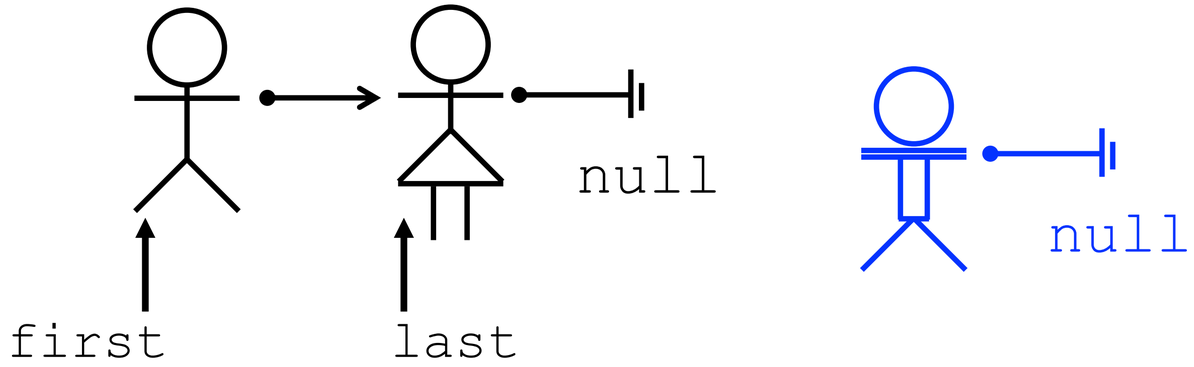
perché?

B.





ulteriore inserimento



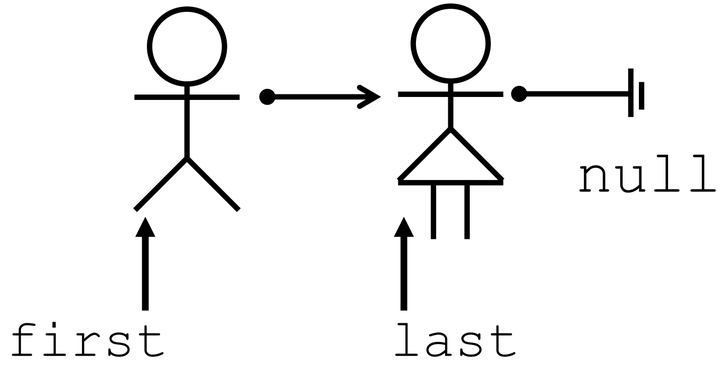

```
newCliente = new
ClientePosta("PaoloBianchi", null)
```

- consideriamo (istanziazione e) accodamento di un altro cliente

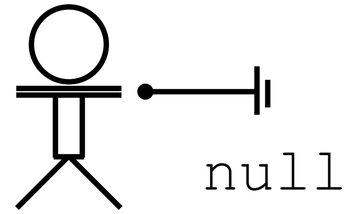
ulteriore inserimento

- consideriamo (istanziazione e) accodamento di un altro cliente

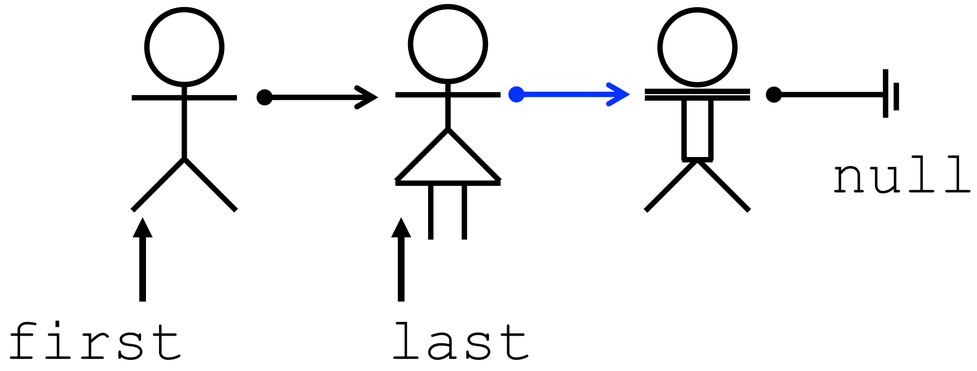
```
last.next = newCliente;
```



coda esistente



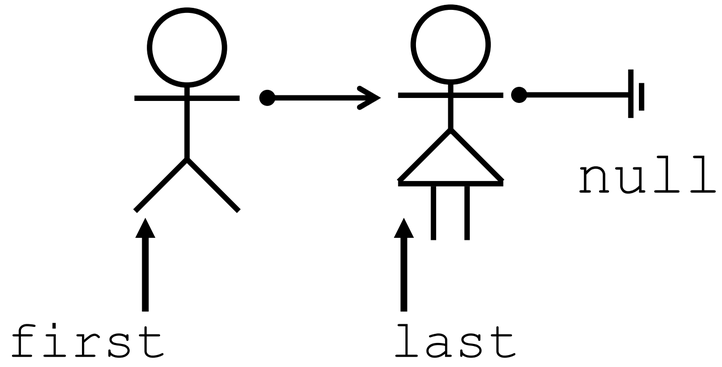
nuovo elem.

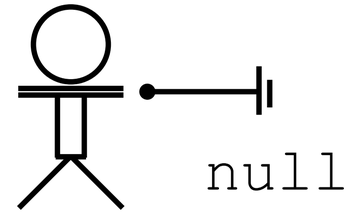


ulteriore inserimento

- consideriamo (istanziazione e) accodamento di un altro cliente

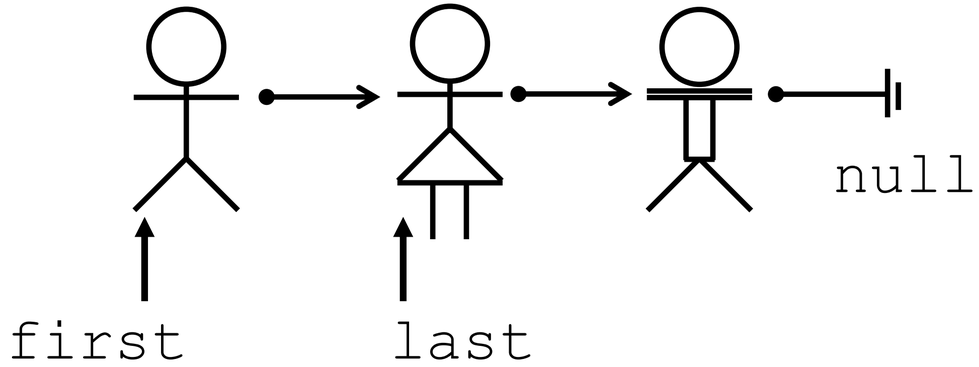
```
last.next = newCliente;

last = newCliente;
```

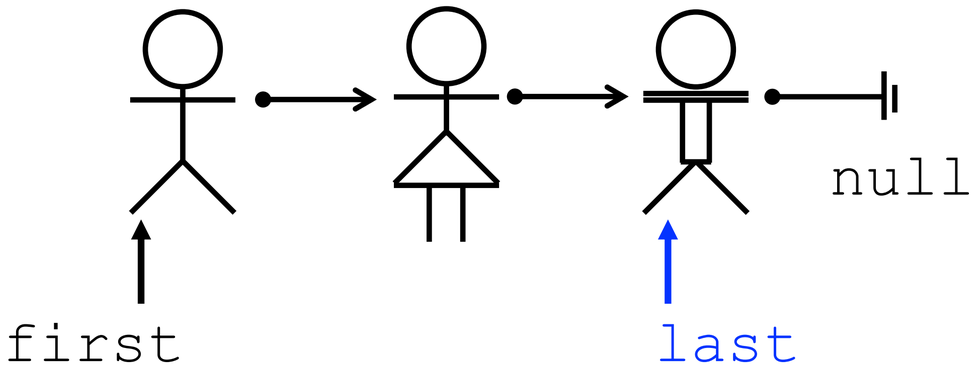




coda esistente

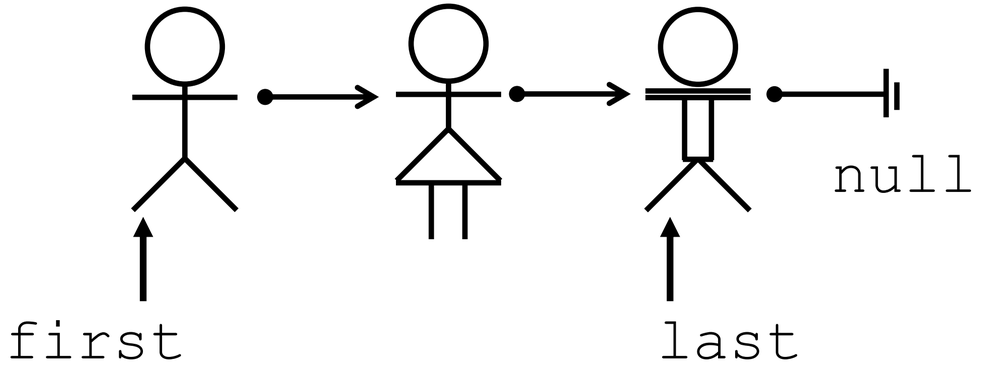
nuovo elem.



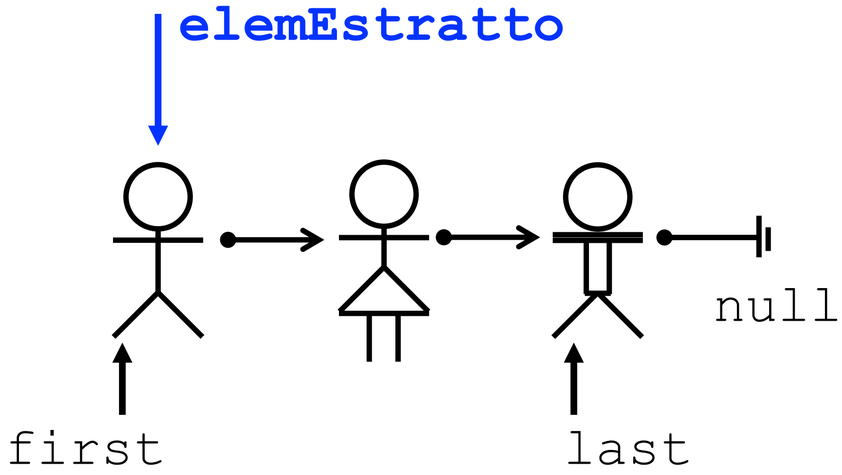


estrazione

- FIFO: si estrae dalla testa, cioè l'elemento riferito dal first:



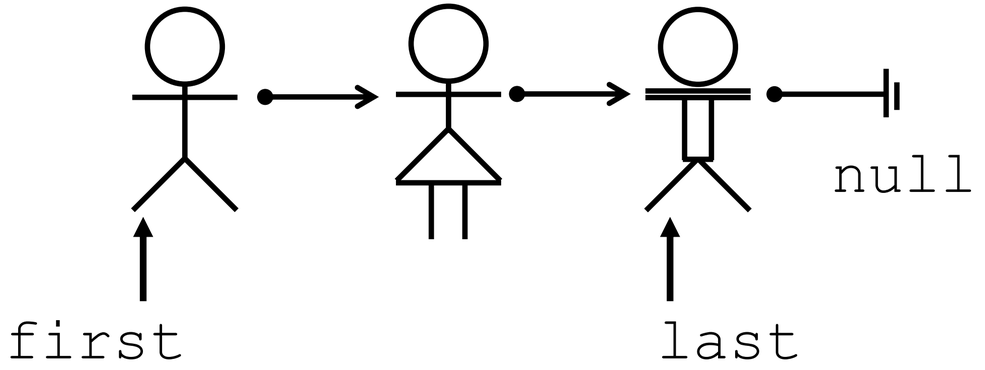

```
ClientePosta elemEstratto = first;
```

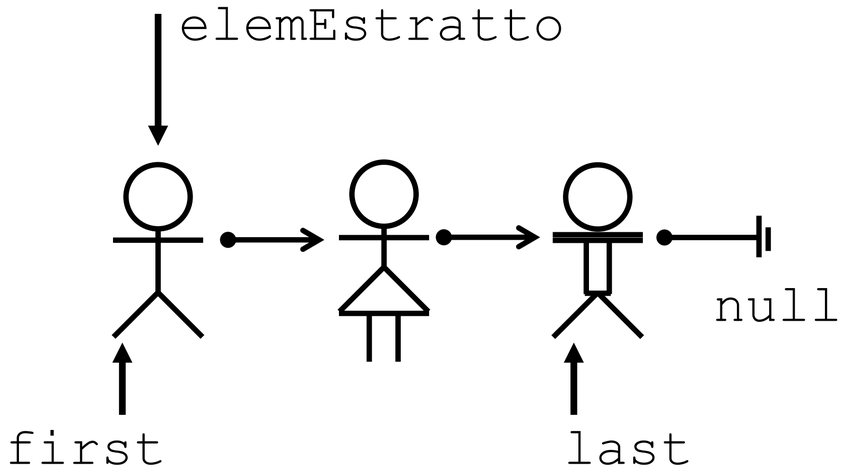


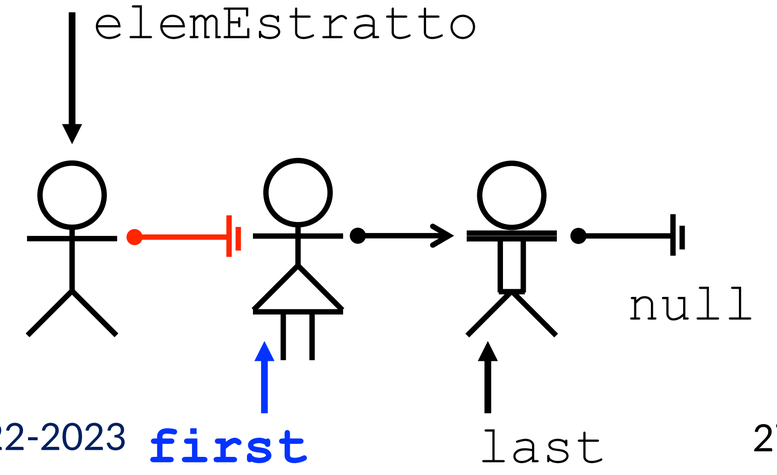
estrazione

- FIFO: si estrae dalla testa, cioè l'elemento riferito dal first:

```
 ClientePosta elemEstratto =
first;
 first = first.next;
 elemEstratto.next = null;
 return elemEstratto;
```



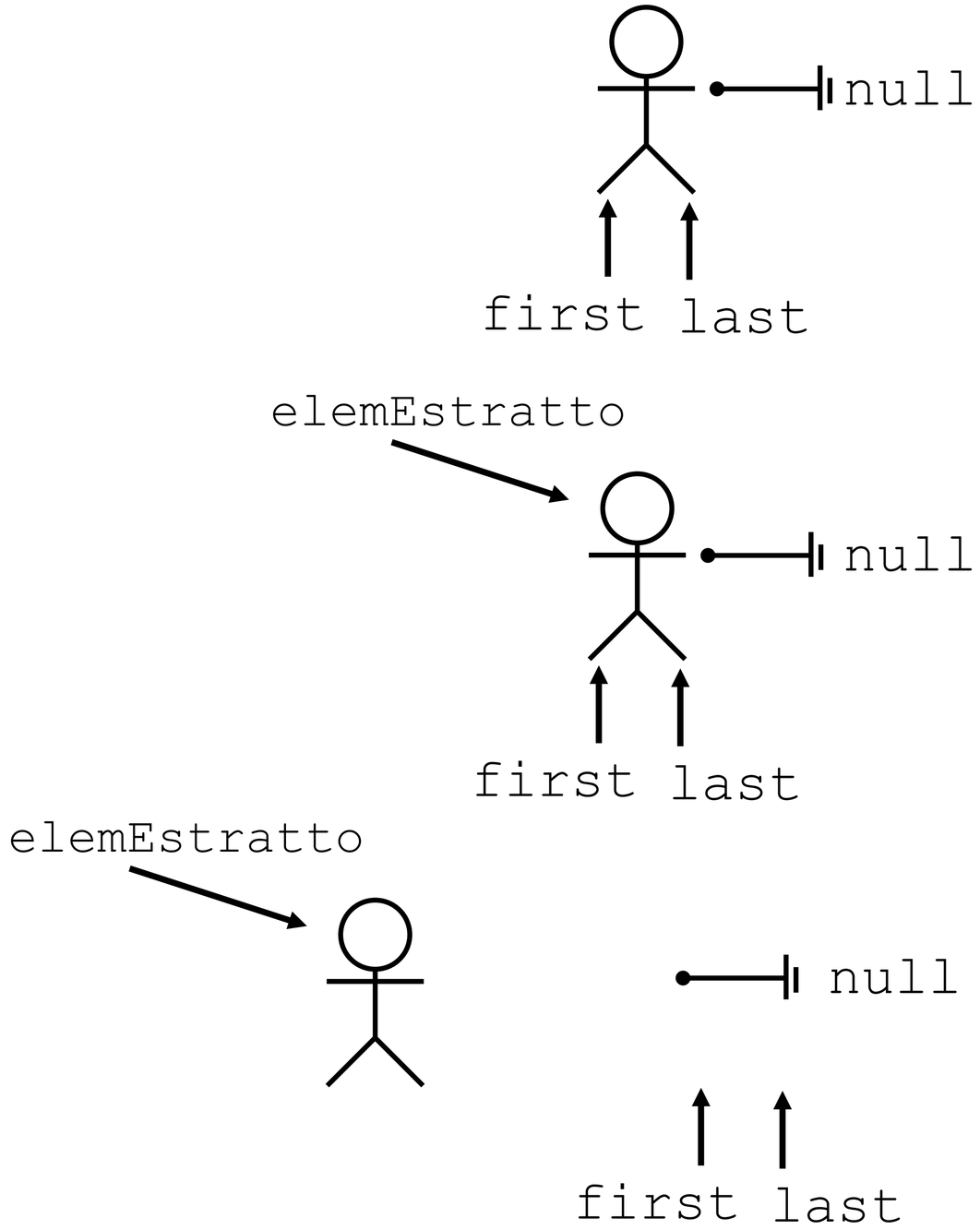




estrazione ultimo

- FIFO: si estrae dalla testa, cioè l'elemento riferito dal first:

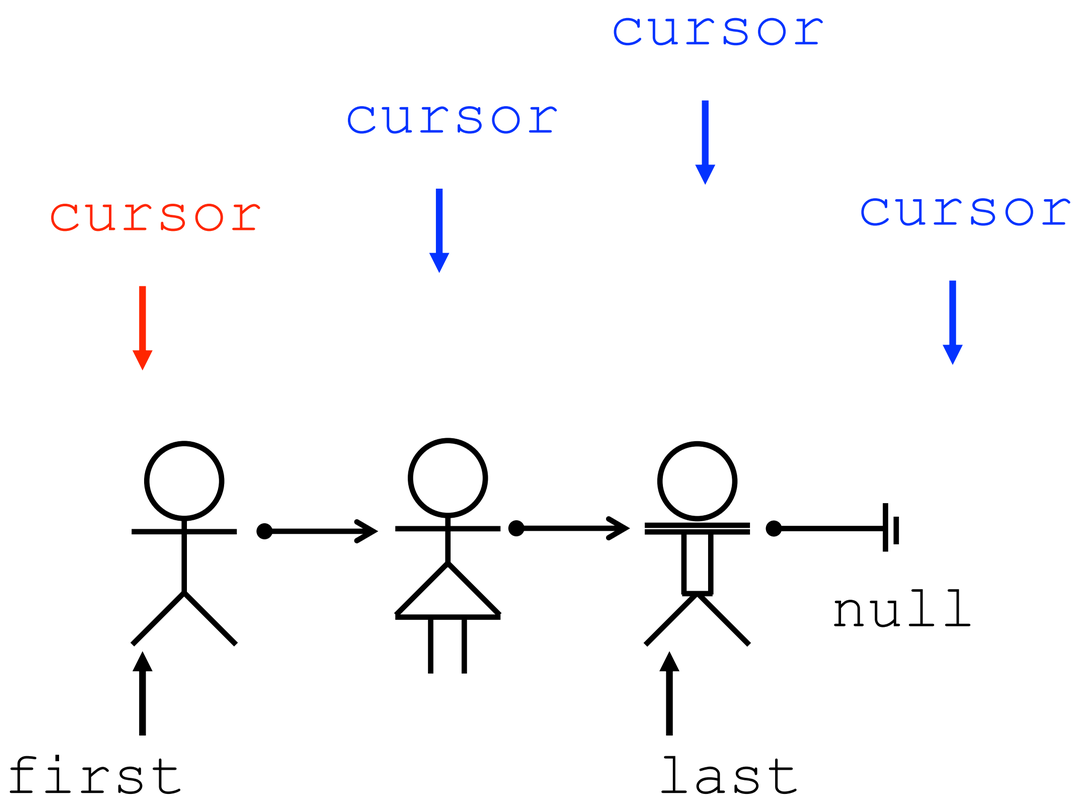
```
 ClientePosta elemEstratto =
first;
 first = first.next;
 elemEstratto.next = null;
 last=null;
 return elemEstratto;
```



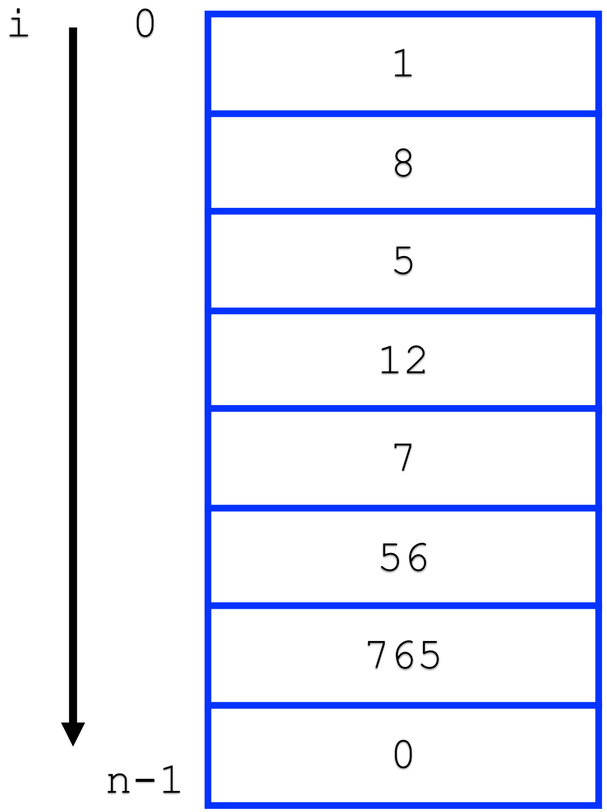
sintesi delle due operazioni

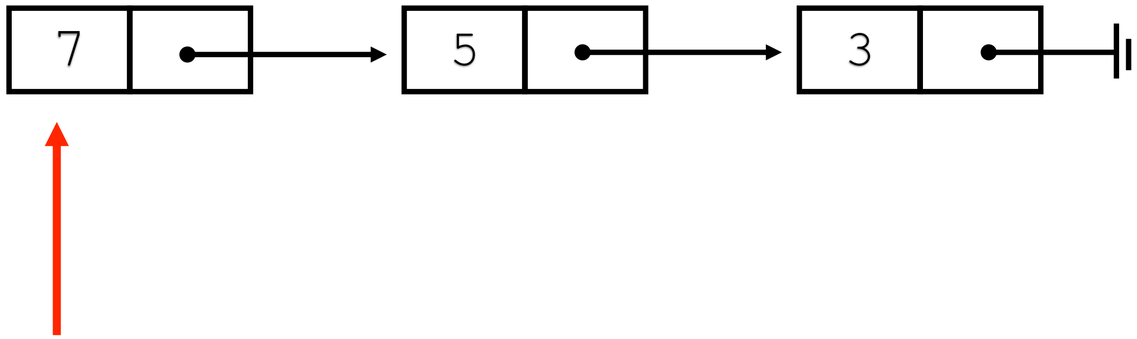
- inserimento in coda
  - creazione nuovoElemento
  - last.next = nuovoElemento
  - last = nuovoElemento

- estrazione da testa
  - elemEstratto = first
  - first = first.next
  - return elemEstratto




```
ClientePosta cursore = first;
while(cursore != null) {
   ...
   cursore = cursore.next;
}
// ho scorso la coda
```


i
0
1
8
5
12
7
56
765
0
n-1


7
5
3

Fine