En classe de teoria s'ha estudiat la representació enllaçada com mecanisme d'emmagatzemament de llistes o seqüències de dades en memòria alternatiu als arrays. En realitat són *Llistes Simplement Enllaçades*, ja que donat un node només és possible accedir al node següent. Per tal de permetre avançar tant al node següent com al node anterior, es planteja construir una *Llista Doblement Enllaçada*, de manera que cada node tinga una referència tant al node següent com al node anterior. La classe `NodeIntDouble` permet representar un node d'una *Llista Doblement Enllaçada*, emmagatzemant una dada de tipus `int`:

```
package linear;
public class NodeIntDouble {
    int data;   // dada del node
    NodeIntDouble next;    // enllaç al següent node
    NodeIntDouble prev;    // enllaç al node anterior

    /** Crea un nou node amb una dada d i que no té
     *  ni anterior ni següent.
     *  @param d int que representa la dada del nou node.
     */
    NodeIntDouble(int d) {
        this(d, null, null);
    }

    /** Crea un nou node amb una dada d, enllaçat a
     *  dos nodes preexistents.
     *  @param d int que representa la dada del nou node.
     *  @param s NodeIntDouble que serà el següent del nou node.
     *  @param a NodeIntDouble que serà l'anterior del nou node.
     */
    NodeIntDouble(int d, NodeIntDouble s, NodeIntDouble a) {
        data = d;  next = s;  prev = a;
    }
}
```

A la Figura 1 es mostren dos exemples d'ús dels constructors de la classe `NodeIntDouble`.

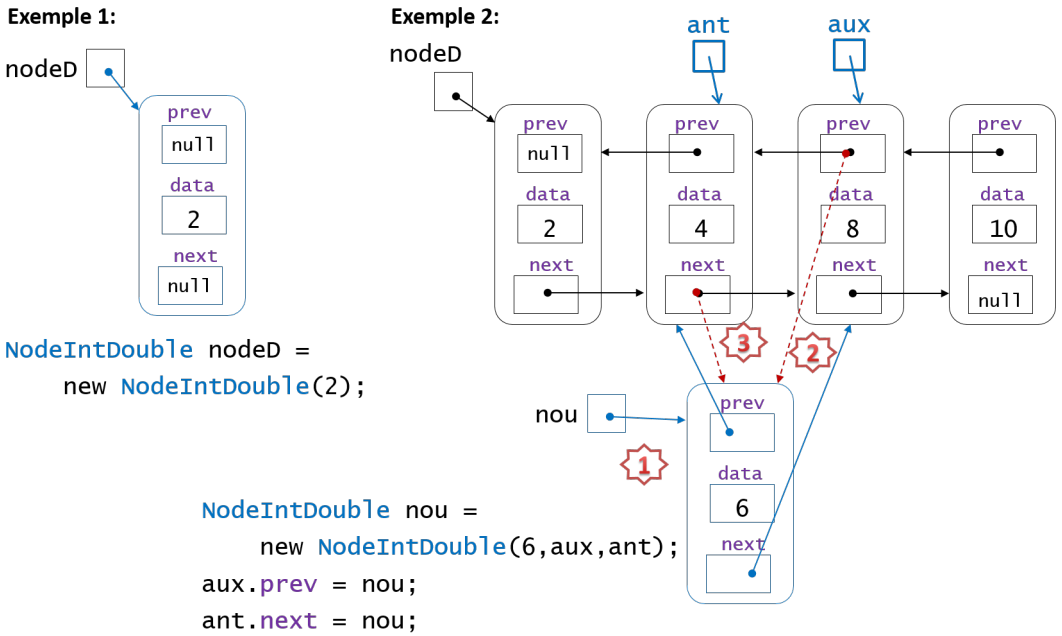


Figura 1: Exemples d'ús dels constructors de `NodeIntDouble`.

Es demana:

1. Descarregar des de PoliformaT les classes `NodeIntDouble`, `ListIntDoublyLinked` i `TestListIntDoublyLinked`[1] en la llibreria d'usuari `linear`.

2. Completar els mètodes de la classe `ListIntDoublyLinked` que té com atributs:

   - `first`, una referència al primer node de la llista.
   - `last`, una referència a l'últim node de la llista.
   - `size`, un enter que represente la talla o número d'elements de la llista.

   A la figura 2 es mostra gràficament la llista `2 4 8 10` com una `ListIntDoublyLinked`.

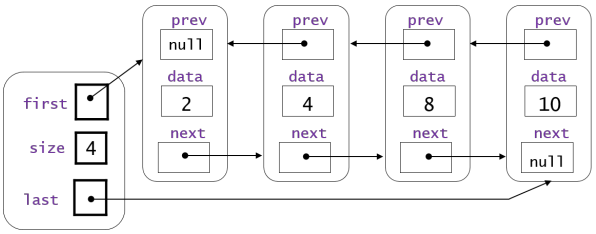


Figura 2: Exemple de `ListIntDoublyLinked`.

Els mètodes a completar són:

1. Constructor per inicialitzar la llista, és a dir, per crear una llista buida (com es mostra en l'exemple de la Figura 3).

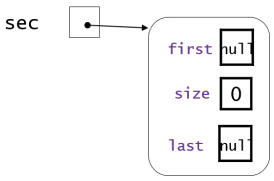
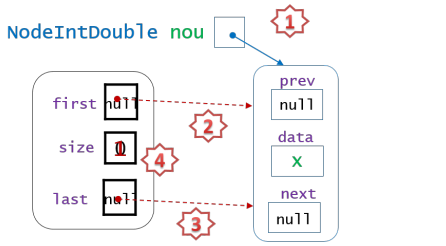


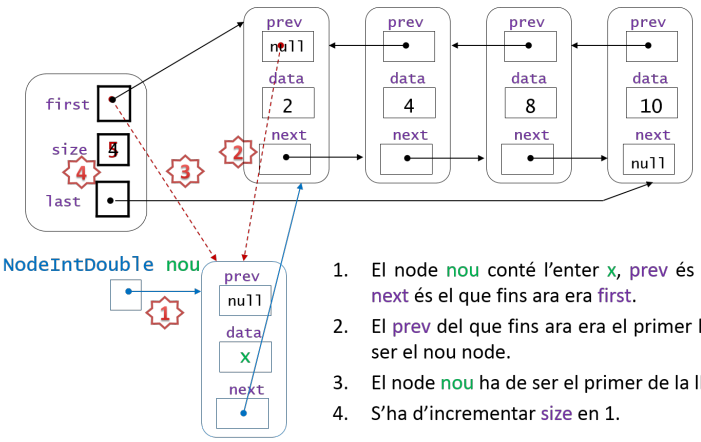
Figura 3: Constructor.

2. Mètode `inserir(int)`, que permeta inserir un nou node en el cap de la llista. En la Figura 4 es mostren els casos a tenir en compte i el que s'ha de fer en cada cas per inserir el nou node en el cap de la llista.



1. El node nou conté l'enter x, prev és null i next és el que fins ara era first, en aquest cas null.
2. El node nou ha de ser el primer i
3. l'últim de la llista.
4. S'ha d'incrementar size en 1.

(a) Cas 1: la llista està buida



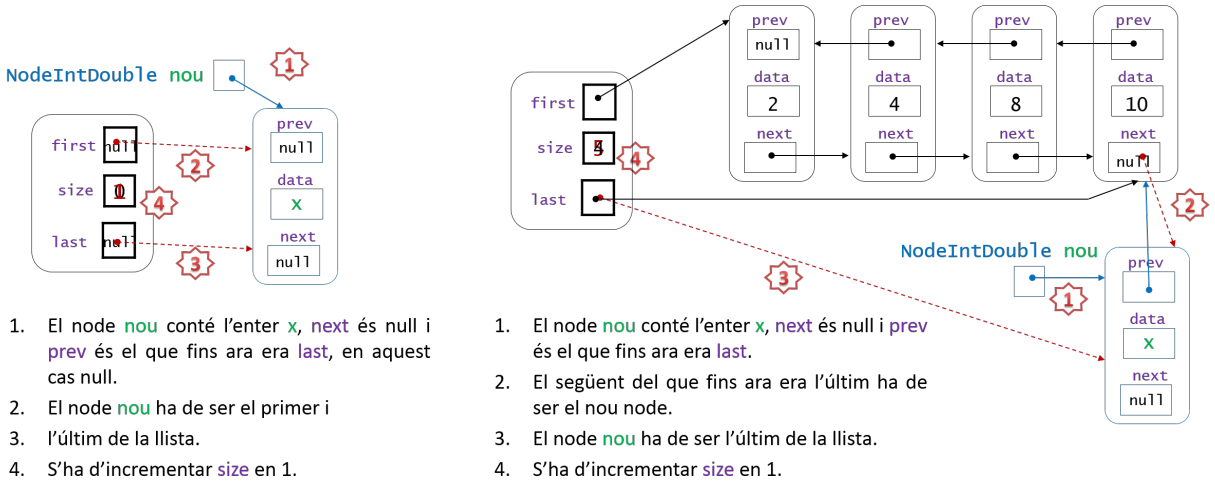
1. El node nou conté l'enter x, prev és null i next és el que fins ara era first.
2. El prev del que fins ara era el primer ha de ser el nou node.
3. El node nou ha de ser el primer de la llista.
4. S'ha d'incrementar size en 1.

(b) Cas 2: la llista NO està buida

Figura 4: Mètode `inserir(int)`.

[1] L'execució del test `TestListIntDoublyLinked` comprova que els distints mètodes de la classe `ListIntDoublyLinked` a implementar siguen correctes.

3. Mètode `inserirFi(int)`, que permeta inserir un nou node al final de la llista. En la Figura 5 es mostren els casos a tenir en compte i el que s'ha de fer en cada cas per inserir el nou node al final de la llista.



(a) Cas 1: la llista està buida

(b) Cas 2: la llista NO està buida

Figura 5: Mètode `inserirFi(int)`.

4. Mètode `toString()`, que permeta obtenir un `String` amb totes les dades de la llista en sentit ascendent (des del primer node de la llista fins l'últim). Si la llista està buida, torna la cadena buida. Es mostra un exemple d'ús d'aquest mètode en la Figura 6.

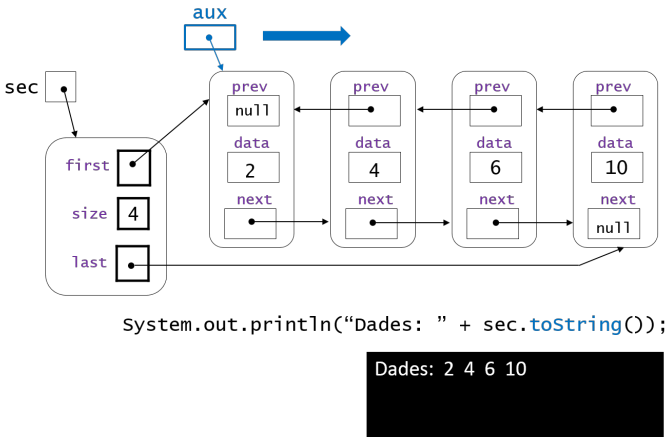


Figura 6: Mètode `toString()`: exemple d'ús.

5. Mètode `toStringDescendent()`, que permeta obtenir un `String` amb totes les dades de la llista en sentit descendent (des de l'últim node de la llista fins el primer). Si la llista està buida, torna la cadena buida. Es mostra un exemple Es mostra un exemple d'ús d'aquest mètode en la Figura 7.

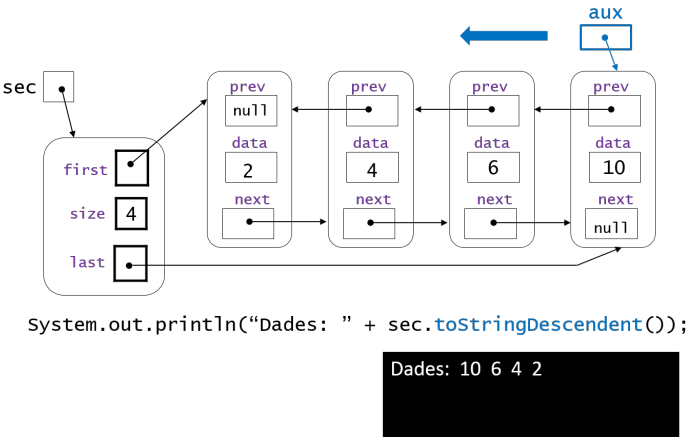
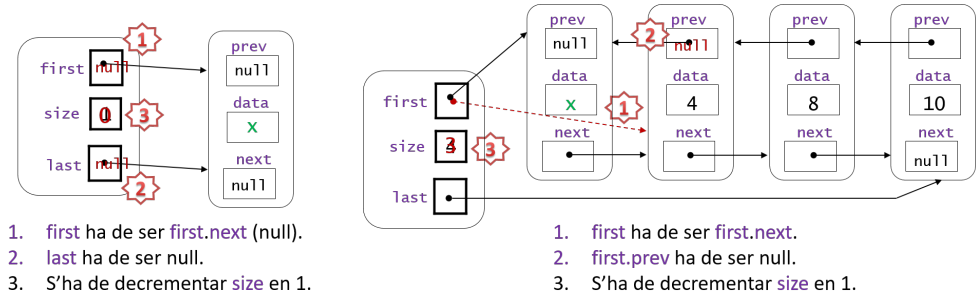


Figura 7: Mètode `toStringDescendent()`: exemple d'ús.

6. Mètode `esborrarIni()`, que esborra, si existeix, el primer node de la llista.
   Si la llista està buida (`first == null`), no s'ha de fer res. Si no està buida, les accions a realitzar seran les que es mostren a la Figura 8. Fixa't que les accions 1 i 3 coincideixen quan la llista té un element i quan té més d'un element.



1. first ha de ser first.next (null).
2. last ha de ser null.
3. S'ha de decrementar size en 1.

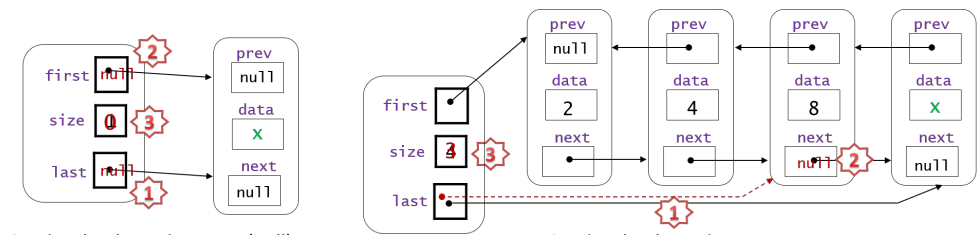
(a) Cas A: llista amb un element

1. first ha de ser first.next.
2. first.prev ha de ser null.
3. S'ha de decrementar size en 1.

(b) Cas B: llista amb més d'un element

Figura 8: Mètode `esborrarIni()`: la llista NO està buida.

7. Mètode `esborrarFi()`, que esborra, si existeix, el darrer node de la llista.
   Si la llista està buida (`last == null`), no s'ha de fer res. Si no està buida, les accions a realitzar seran les que es mostren a la Figura 9. Fixa't que les accions 1 i 3 coincideixen quan la llista té un element i quan té més d'un element.



1. last ha de ser last.prev (null).
2. first ha de ser null.
3. S'ha de decrementar size en 1.

(a) Cas A: llista amb un element

1. last ha de ser last.prev.
2. last.next ha de ser null.
3. S'ha de decrementar size en 1.

(b) Cas B: llista amb més d'un element

Figura 9: Mètode `esborrarFi()`: la llista NO està buida.

8. Mètode `esborrar(int)`, que permeta esborrar, si existeix, la primera ocurrència d'una dada donada (Figura 11). Si no existeix, no fa res (Figura 10).
   Per tal d'esborrar una dada donada de la llista, el primer que s'ha de fer és trobar-la. Si no existeix, bé perque la llista està buida, bé perque no es troba, com es mostra a la Figura 10, no s'ha de fer res.

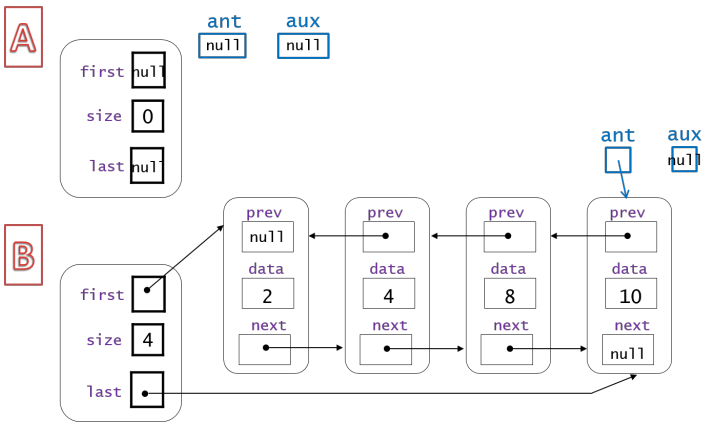
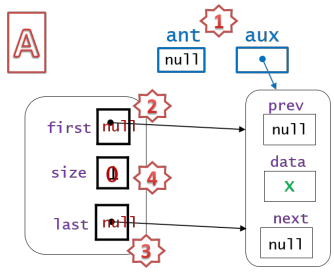


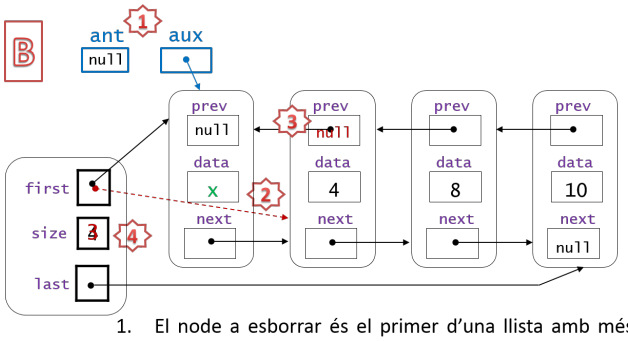
Figura 10: Mètode `esborrar(int)`: la dada a esborrar NO existeix (la llista pot estar buida o no).

Si la dada a esborrar està a la llista, com es mostra a la Figura 11, les accions a realitzar seràn unes o altres en funció de quin siga el node a esborrar. No obstant, hi ha casos que comparteixen certes accions.



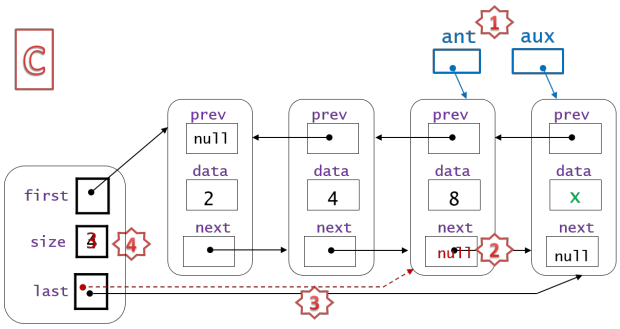
1. El node a esborrar és el primer d'una llista amb un element i, per tant, també és l'últim de la llista.
2. first ha de ser null (aux.next).
3. last ha de ser null (ant).
4. S'ha de decrementar size en 1.

(a) Cas 2A: el node a esborrar és el primer d'una llista amb un element



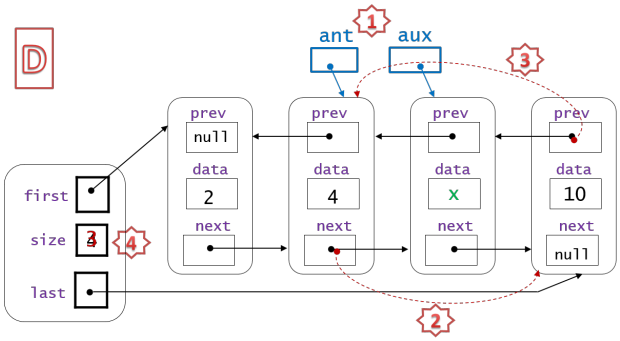
1. El node a esborrar és el primer d'una llista amb més d'un element.
2. first ha de ser aux.next.
3. El prev de aux.next (aux.next.prev) ha de ser null (ant).
4. S'ha de decrementar size en 1.

(b) Cas 2B: el node a esborrar és el primer d'una llista amb més d'un element



1. El node a esborrar és l'últim d'una llista amb més d'un element.
2. ant.next ha de ser null (aux.next).
3. last ha de ser ant.
4. S'ha de decrementar size en 1.

(c) Cas 2C: el node a esborrar és l'últim d'una llista amb més d'un element



1. El node a esborrar no és ni el primer ni l'últim de la llista.
2. ant.next ha de ser aux.next.
3. El prev de aux.next (aux.next.prev) ha de ser ant.
4. S'ha de decrementar size en 1.

(d) Cas 2D: el node a esborrar no és el primer ni l'últim de la llista

Figura 11: Mètode `esborrar(int)`: la dada a esborrar existeix (la llista no està buida).

Nota: El codi ha de ser òptim (per exemple, evitant instruccions redundants o duplicades) i ha de seguir l'estil de codificació recomanat en Java segons el Checkstyle de *BlueJ*.