Listas enlazadas

Programación de Sistemas

Estructuras de datos

- Abstracción que representa un conjunto de datos en un programa con el objeto de facilitar su manipulación
- Diferentes estructuras de datos presentan ventajas o desventajas dependiendo de la naturaleza de los datos y los tipos de manipulaciones que se necesite hacer sobre los mismos

Estructuras de datos lineales

- Organizan los datos en secuencia, donde cada dato se relaciona con un dato anterior (excepto el primero) y un dato posterior (excepto el último)
- Ejemplos de estructuras de datos lineales:
  - Arrays
  - Listas enlazadas
  - Pilas
  - Colas
  - Colas dobles

Arrays

- Los *arrays* presentan dos grandes ventajas para el almacenamiento de colecciones lineales de datos:
  - Acceso aleatorio: se puede acceder a cualquier posición del array en tiempo constante.
  - Uso eficiente de memoria cuando todas las posiciones están ocupadas: por guardarse en posiciones consecutivas de memoria.

Arrays

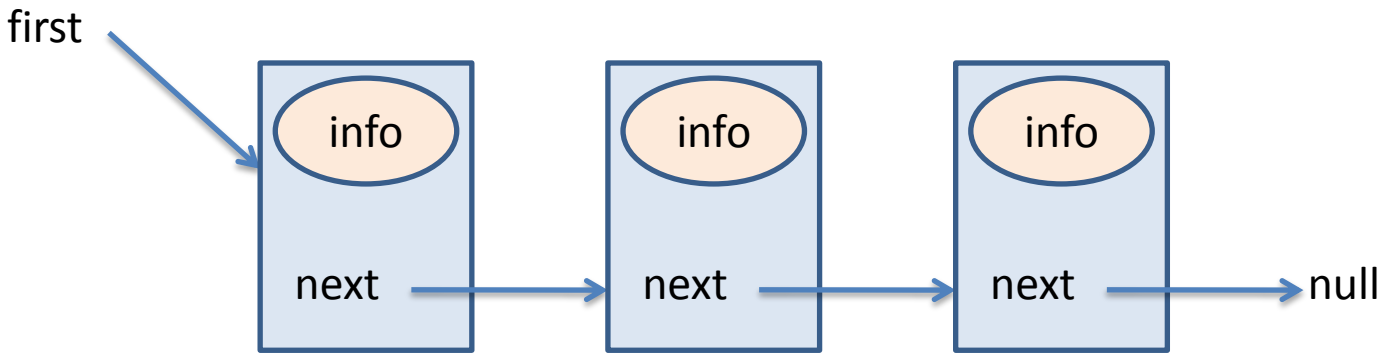
- Desventajas:
  - Tamaño estático: debe asignarse un tamaño al crear el array, y no se puede cambiar. Da lugar a problemas:
    - Uso no eficiente de memoria por tener que reservar espacio para el caso peor
    - Posibilidad de sobrepasar el tamaño reservado en tiempo de ejecución
  - Necesidad de memoria contigua:
    - Puede ocurrir que, pese a haber suficiente memoria libre, no haya un bloque contiguo suficientemente grande

Arrays

- Desventajas:
  - Ciertas operaciones tienen un coste no óptimo:
    - Inserciones y eliminaciones de datos en la primera posición o posiciones intermedias: necesidad de desplazar datos entre posiciones consecutivas.
    - Concatenación de dos o más arrays: necesidad de copiar los datos a un nuevo array.
    - Partición de un array en varios fragmentos: necesidad de copiar datos a nuevos arrays.

Listas enlazadas

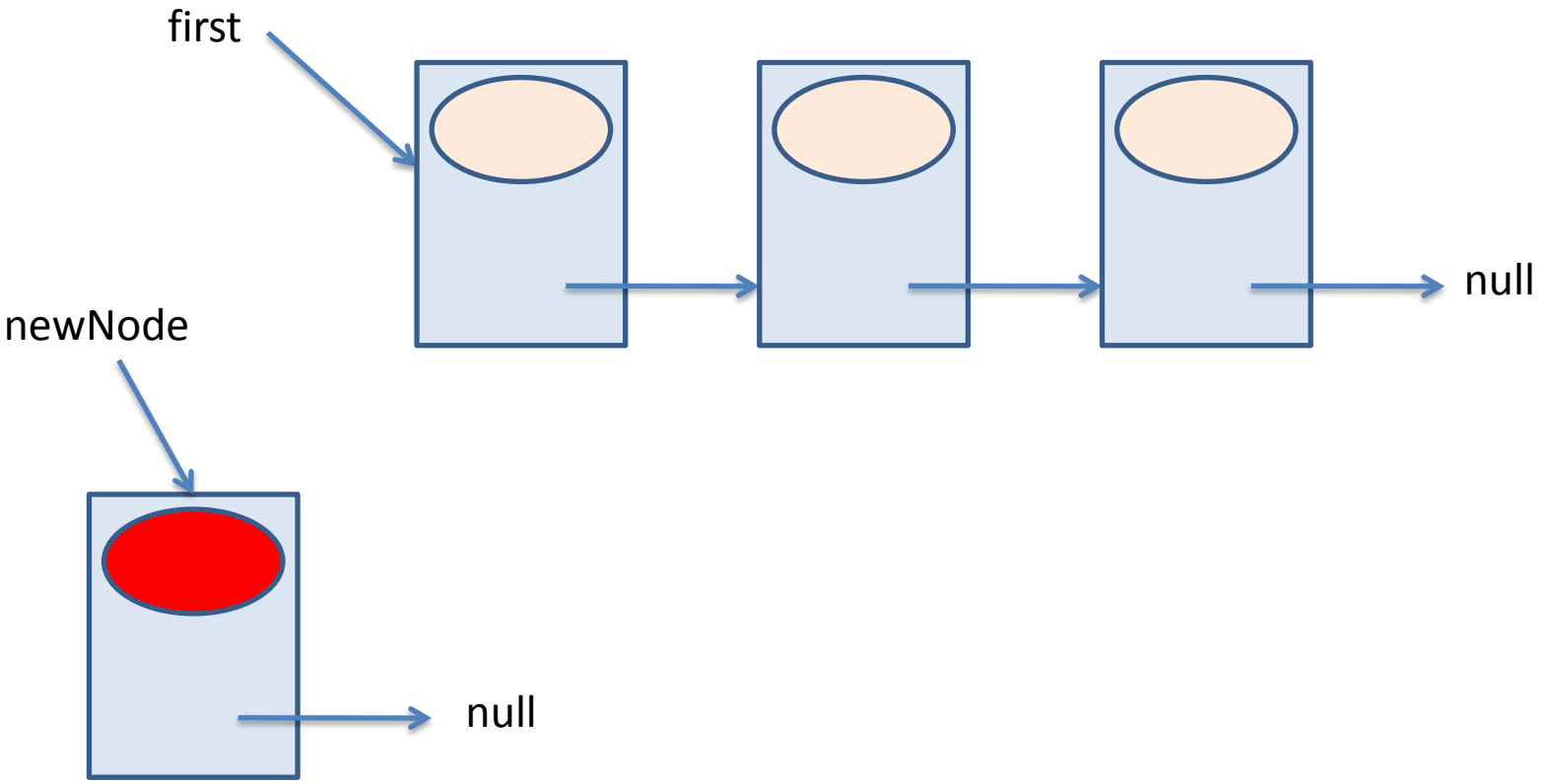
- Secuencia ordenada de nodos donde cada nodo almacena:
  - Un dato
  - Una referencia al siguiente nodo
- Los nodos no tienen por qué estar contiguos en memoria



La clase Node

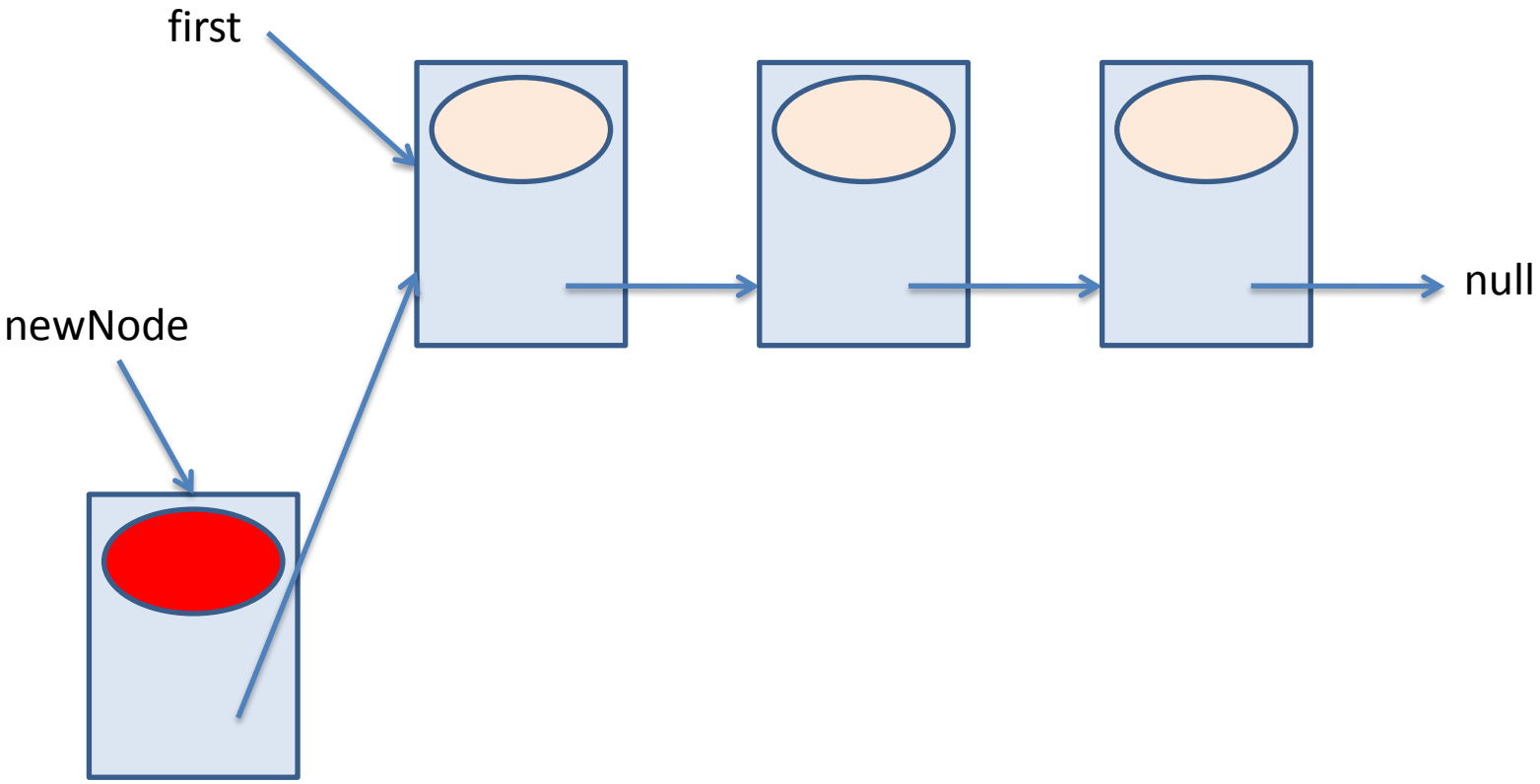
```
Public class Node {
    private Object info;
    private Node next;

    public Node(Object info) {…}

    public Node getNext() {…}
    public void setNext(Node next) {…}
    public Object getInfo() {…}
    public void setInfo(Object info) {…}
}
```

Inserción al principio



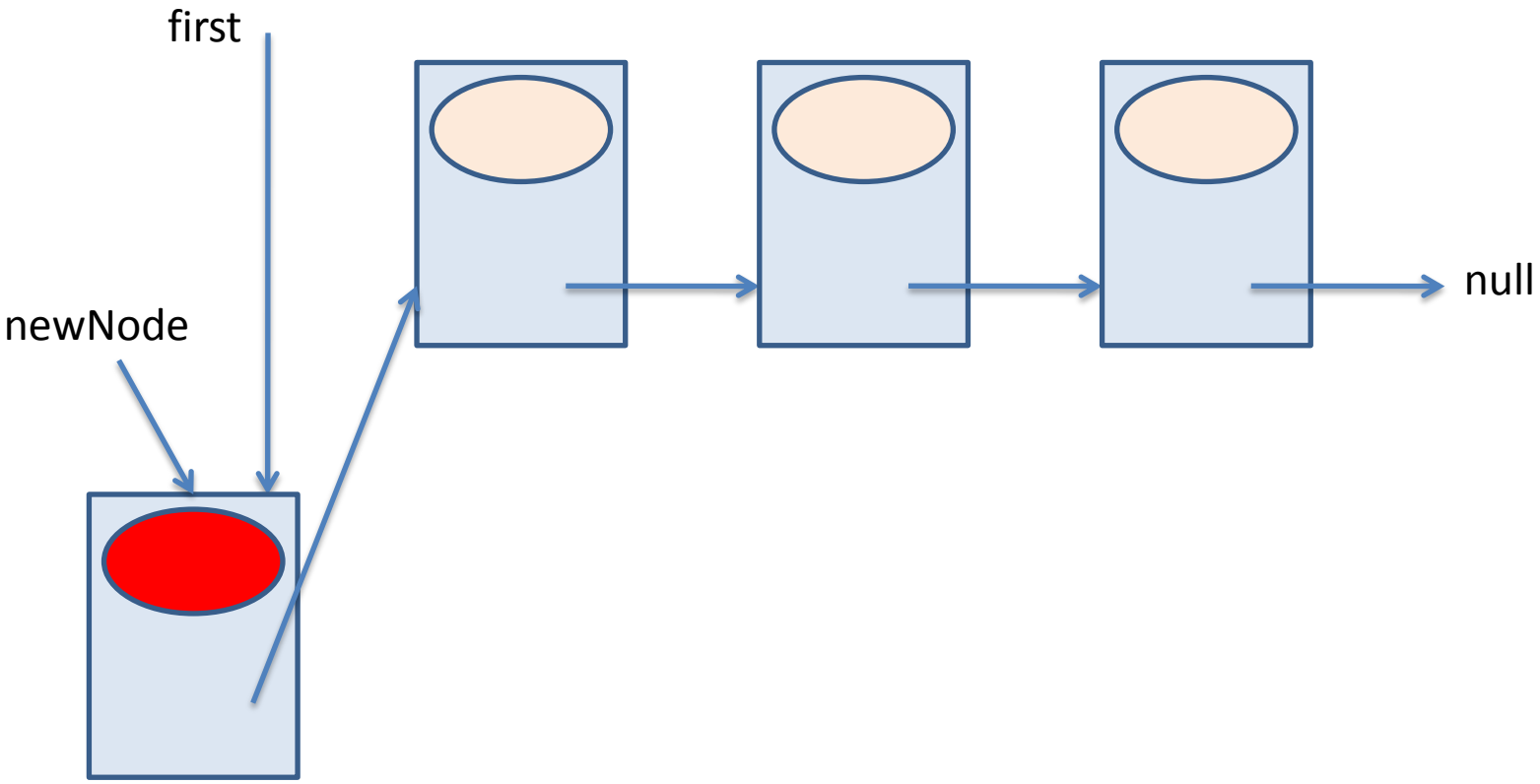
Node newNode = new Node(info);

Inserción al principio



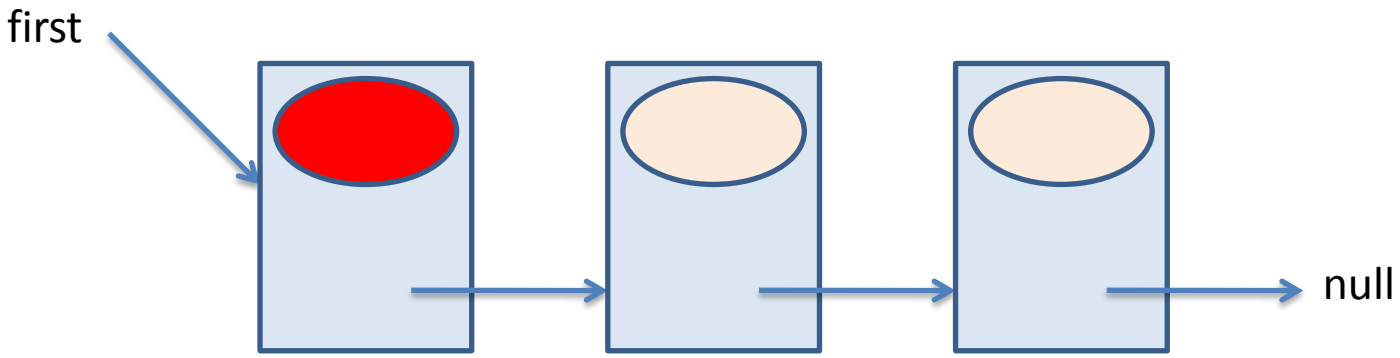
newNode .setNext(first);

Inserción al principio

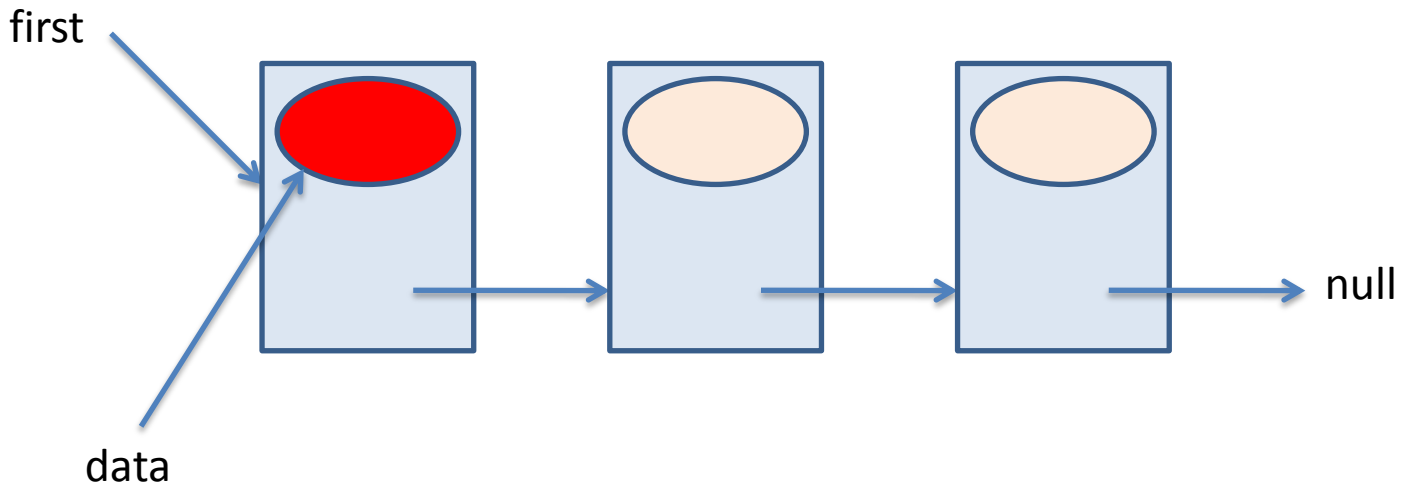


first = newNode;

Extracción del primer nodo

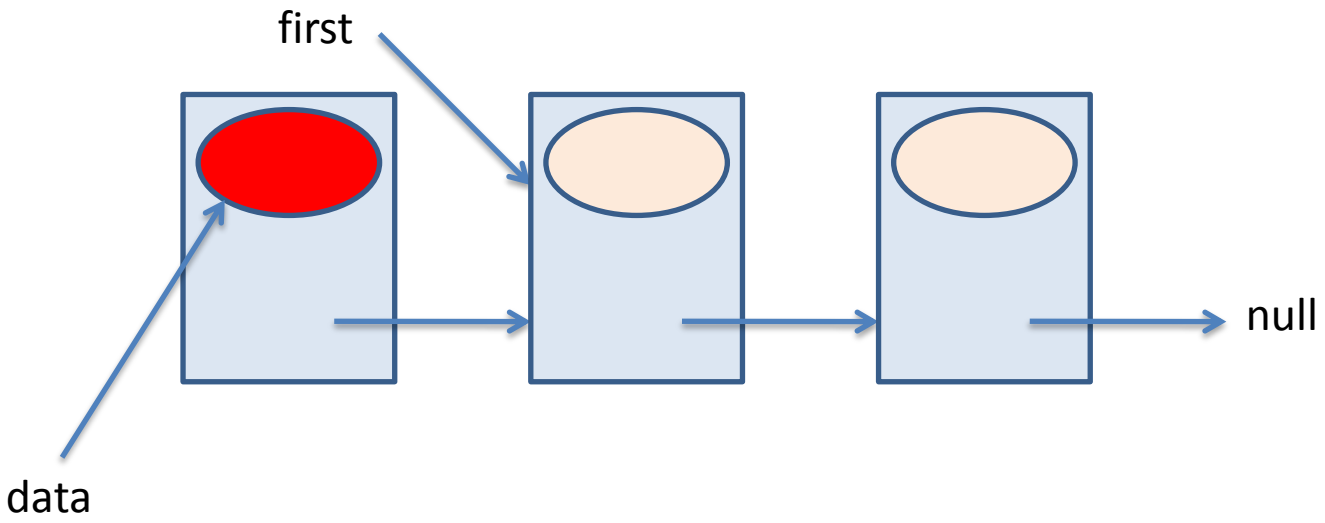


Extracción del primer nodo



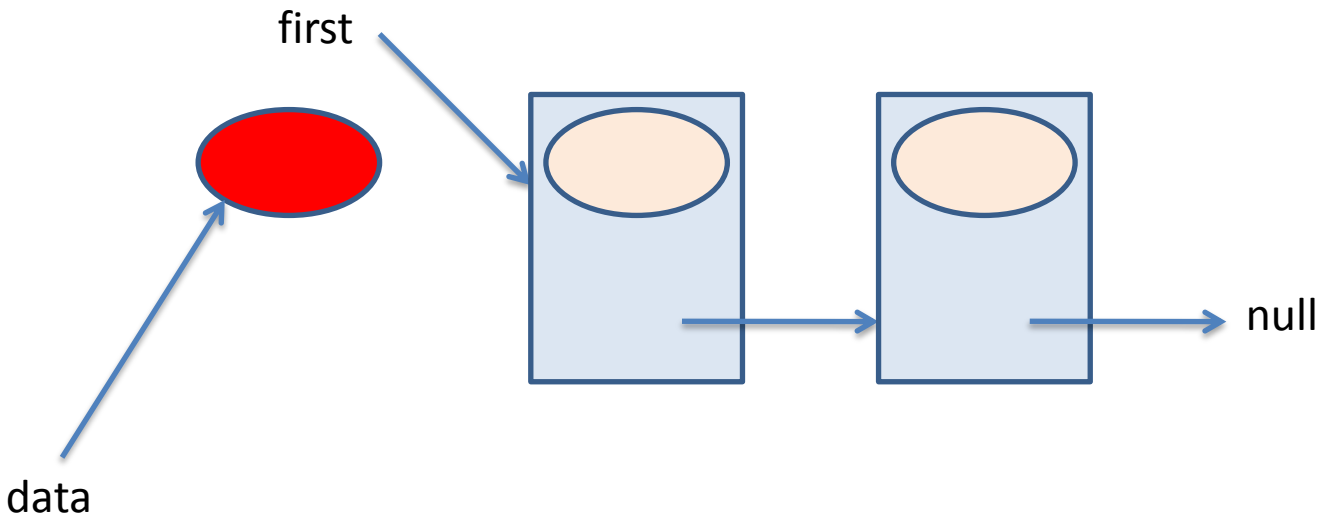
Object data = first.getInfo();

Extracción del primer nodo

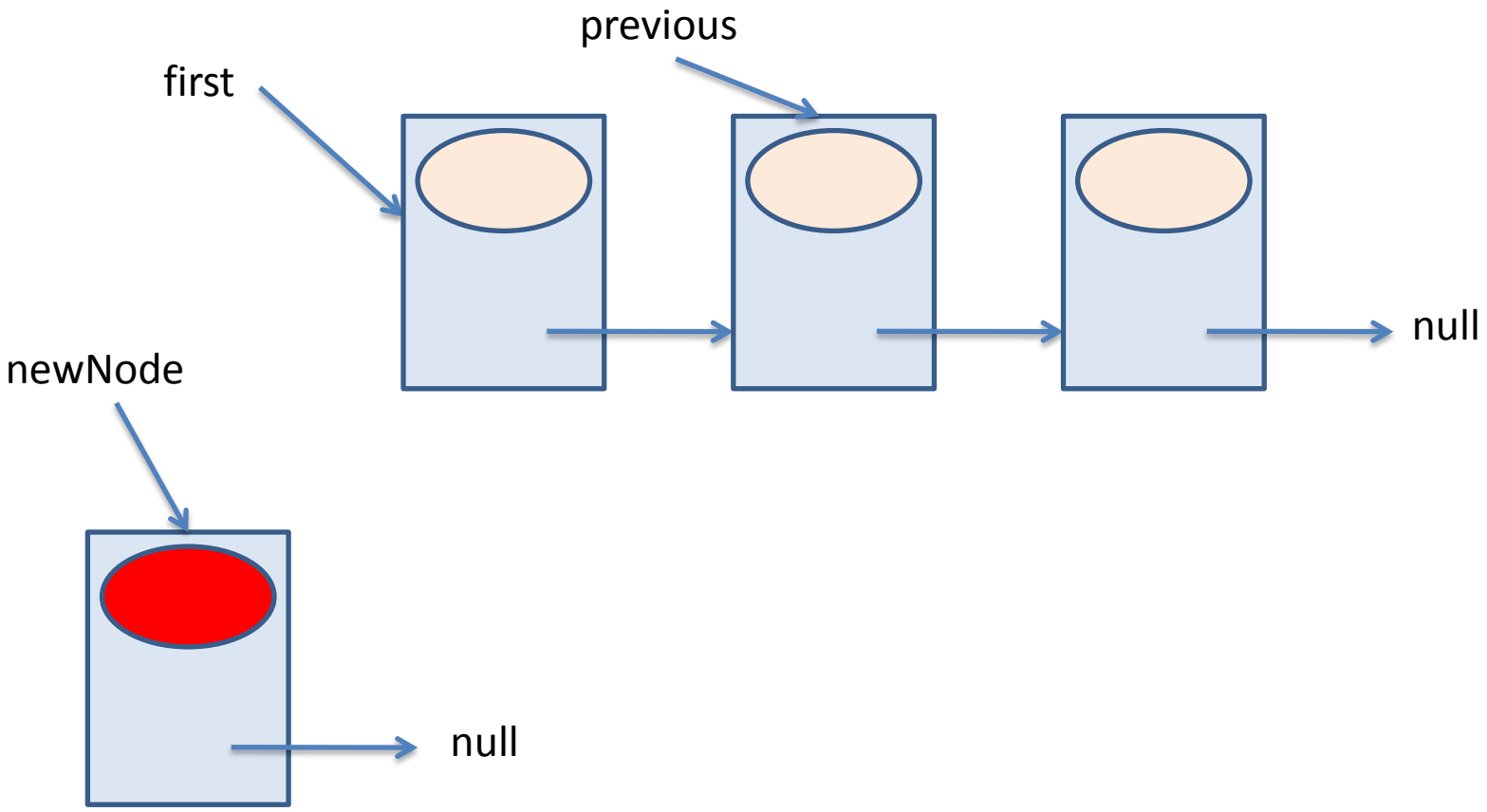


first = first.getNext();

Extracción del primer dato

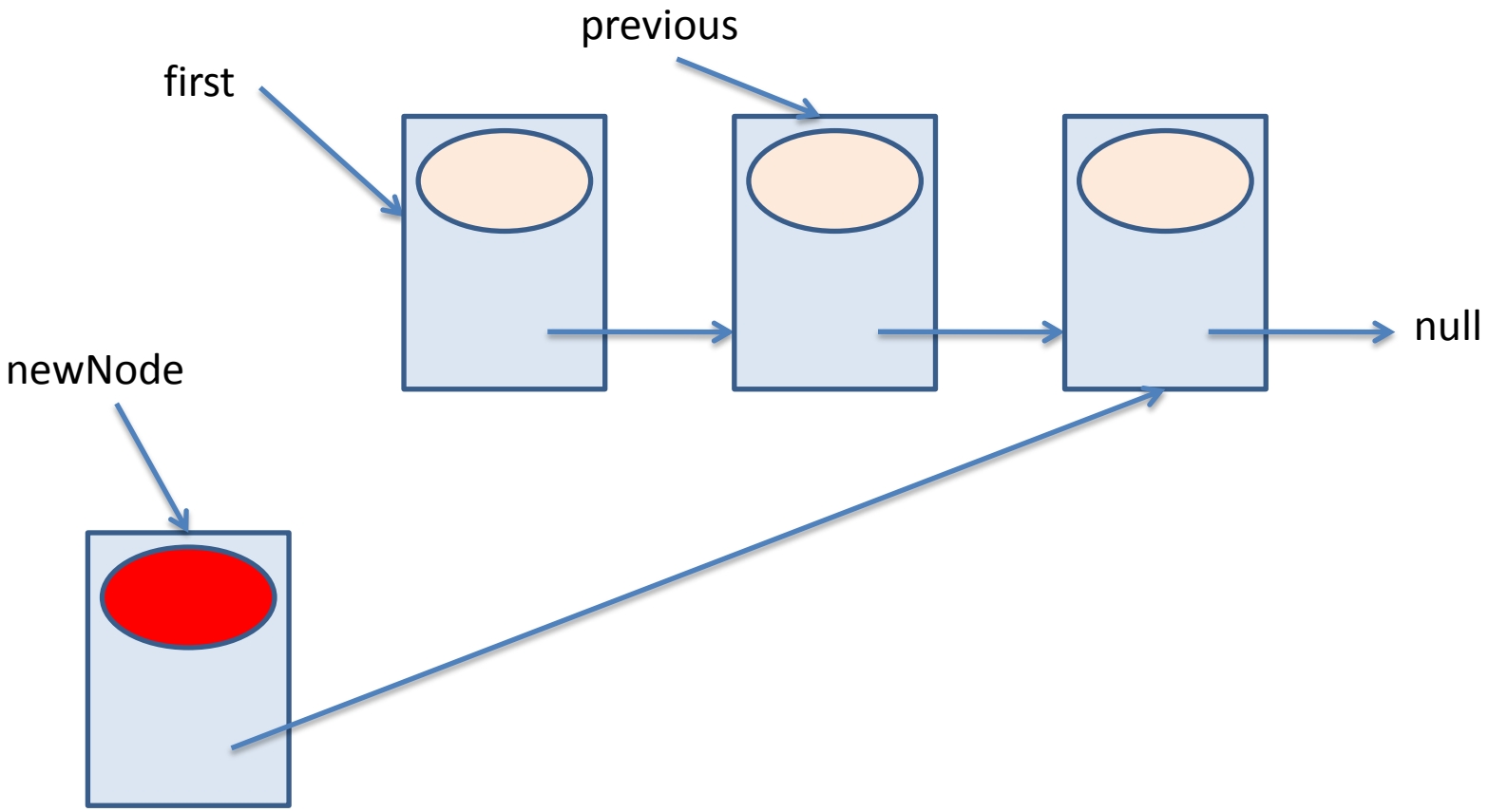


Inserción en un punto intermedio



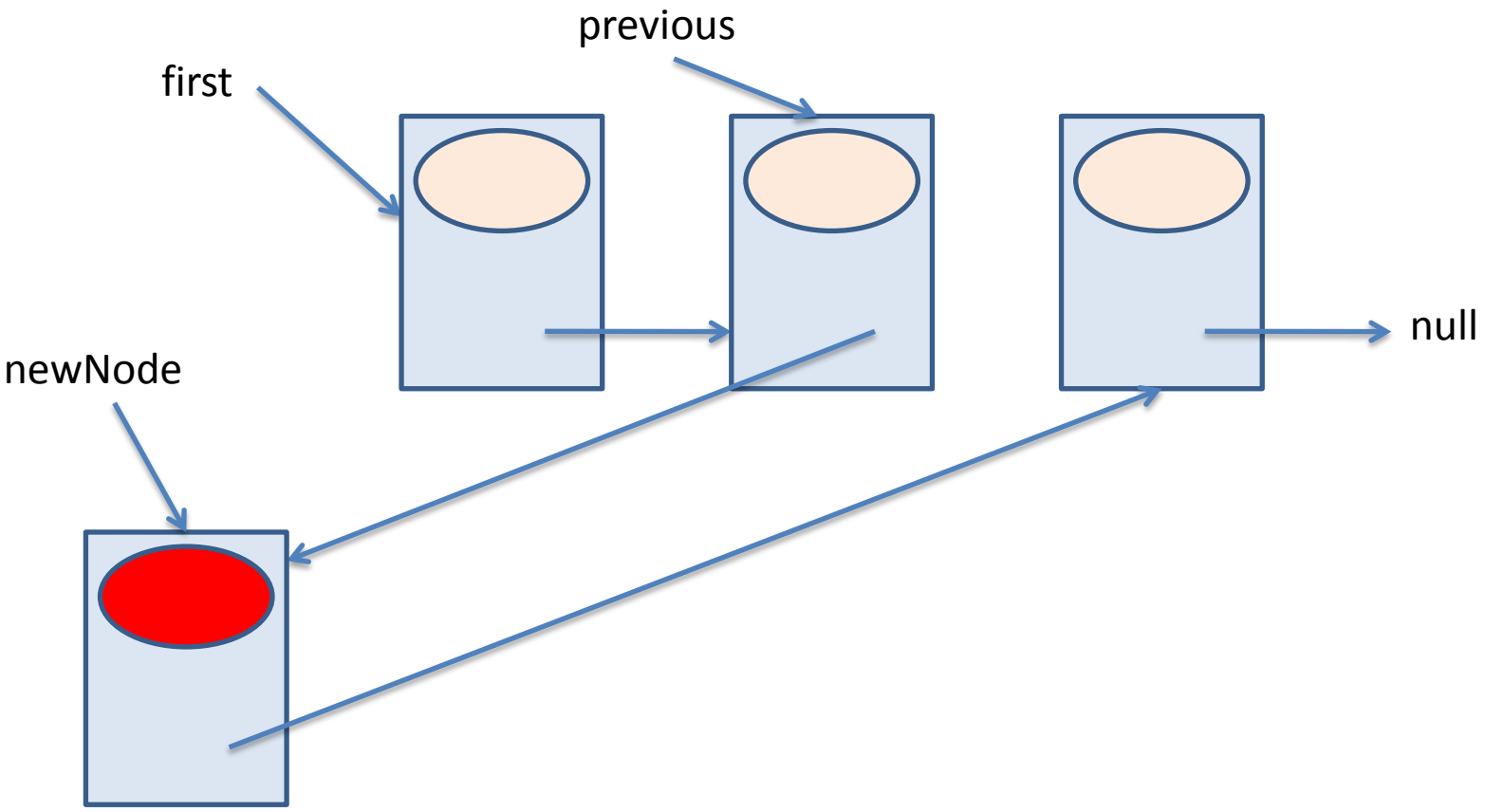
Node newNode = new Node(info);

Inserción en un punto intermedio



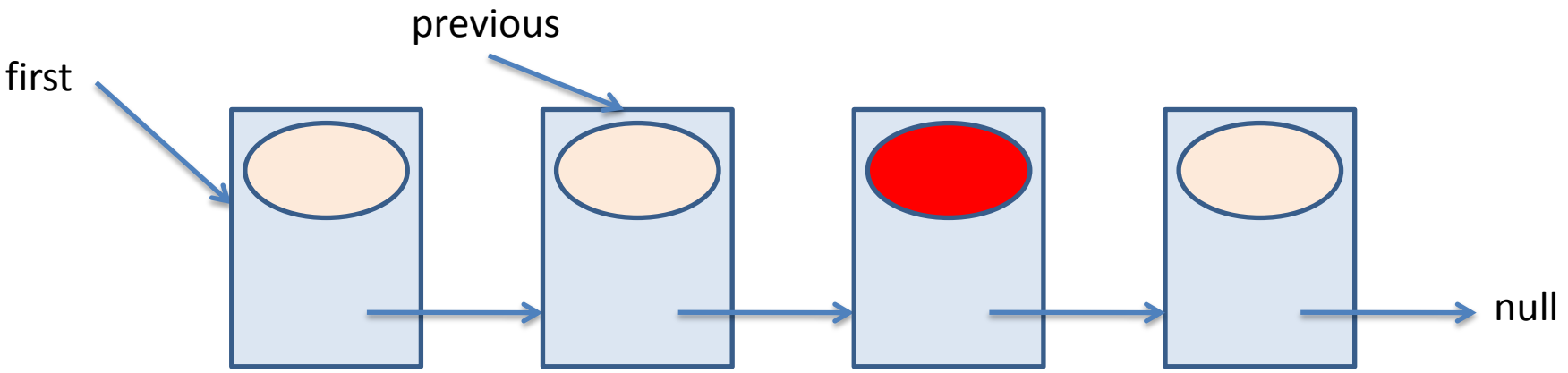
newNode.setNext(previous.getNext())

Inserción en un punto intermedio

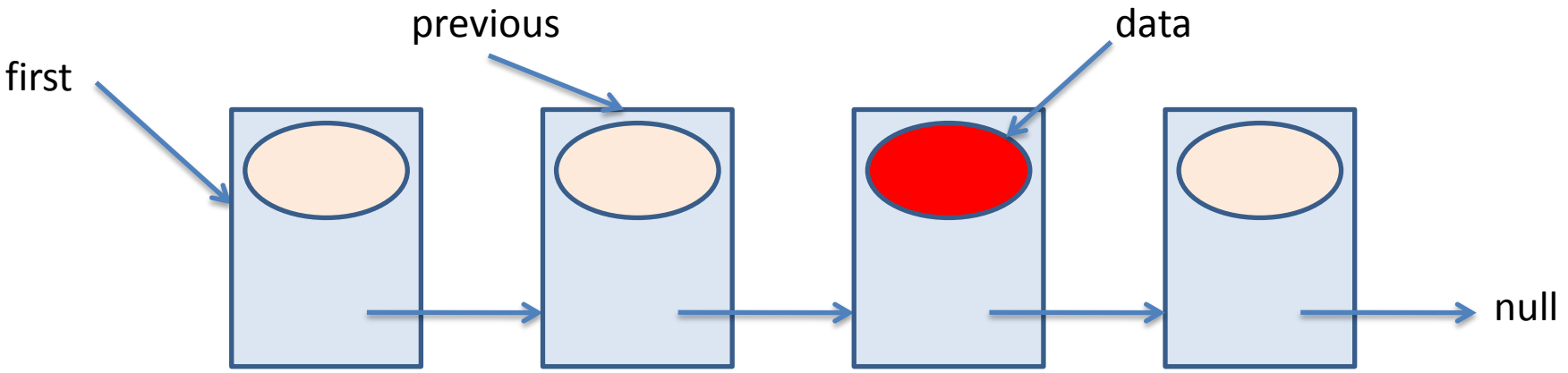


previous.setNext(newNode)

Eliminación de un dato intermedio

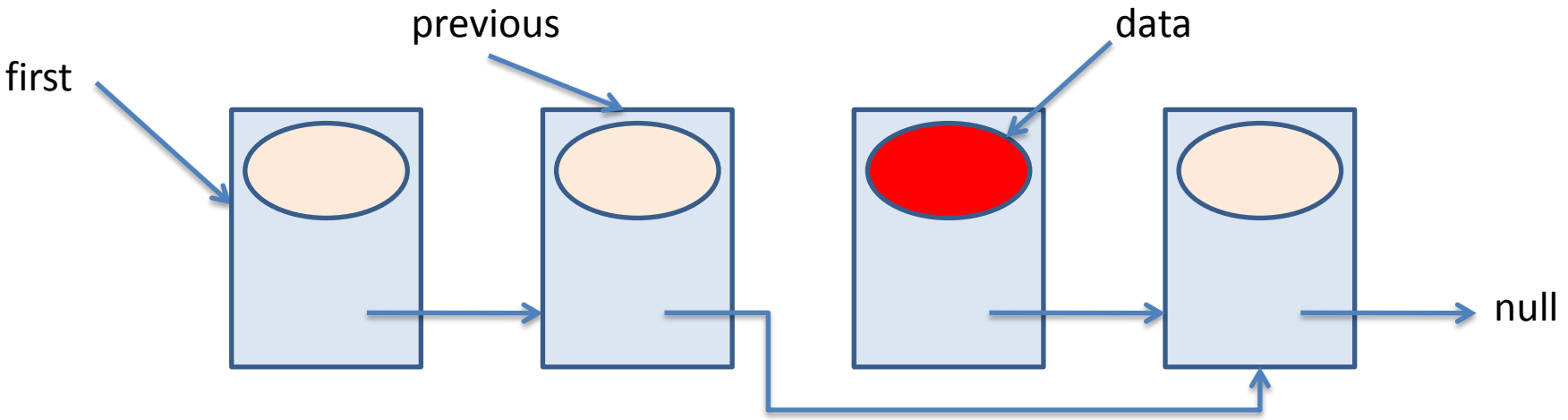


Eliminación de un dato intermedio



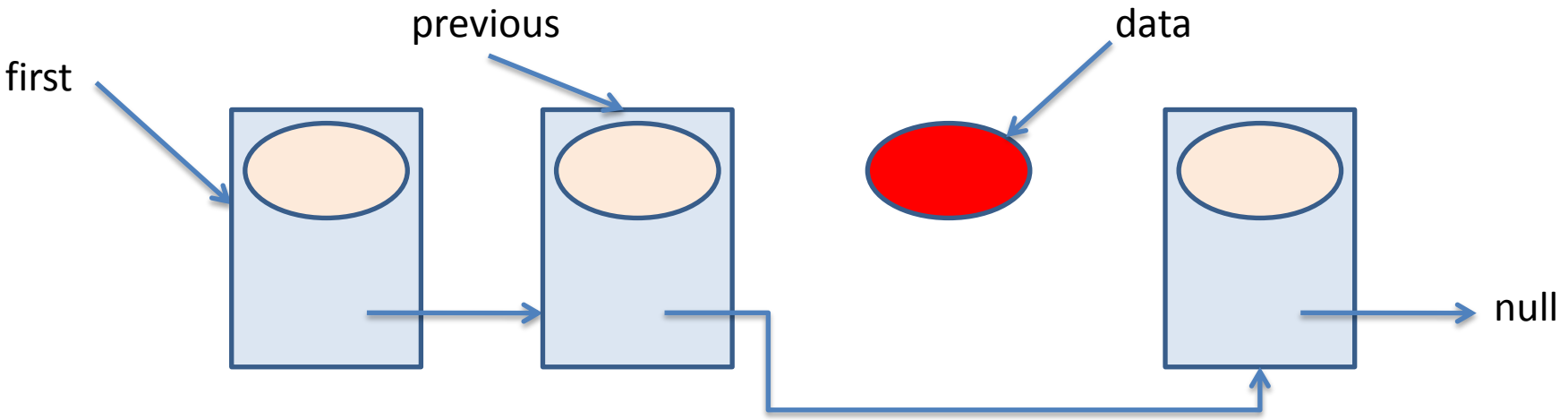
Object data = previous.getNext().getInfo();

Eliminación de un dato intermedio

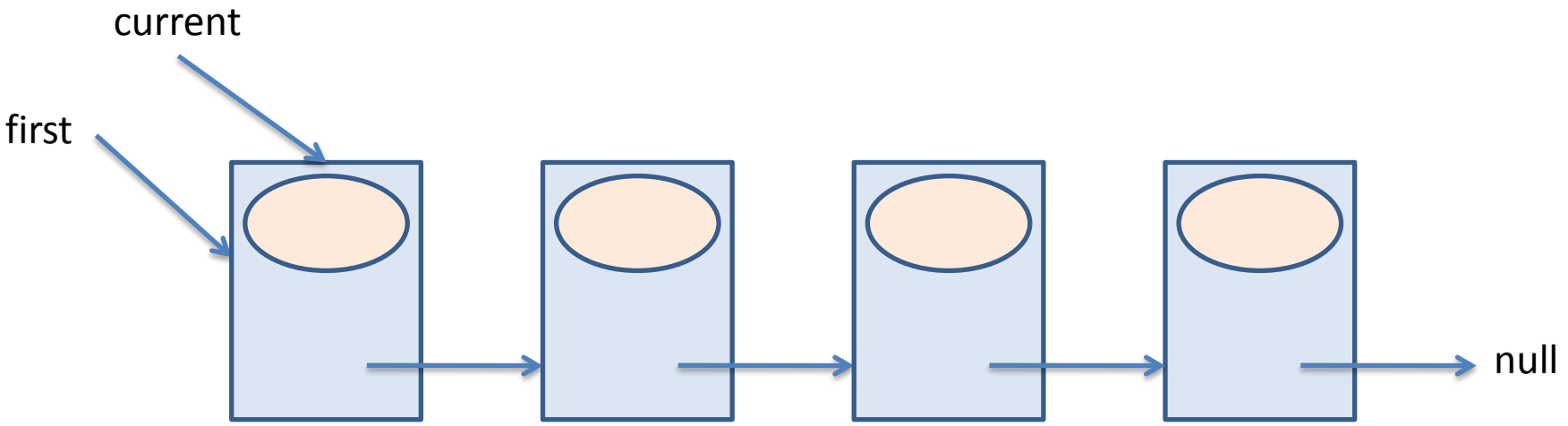


previous.setNext(previous.getNext().getNext())

Eliminación de un dato intermedio



Recorrido de la lista




```
Node current = first;
while (current != null) {
    current = current.getNext();
}
```

Recorrido: buscar el último nodo

- Se avanza una referencia hasta localizar un nodo cuyo siguiente sea null:

```
public Node searchLastNode() {
    Node last = null;
    Node current = first;
    if (current != null) {
        while (current.getNext() != null) {
            current = current.getNext();
        }
        last = current;
    }
    return last;
}
```

Recorrido: buscar posición de un dato

- Se avanza una referencia hasta localizar el dato. Se va incrementando un contador al mismo tiempo:

```
public int search(Object info) {
    int pos = 1;
    Node current = first;
    while (current != null
           && !current.getInfo().equals(info)) {
        pos += 1;
        current = current.getNext();
    }
    if (current != null)
        return pos;
    else
        return -1;
}
```

Ventajas de las listas enlazadas

- Inserción y extracción de nodos con coste independiente del tamaño de la lista
- Concatenación y partición listas con coste independiente del tamaño de las listas
- No hay necesidad de grandes cantidades de memoria contigua
- El uso de memoria se adapta dinámicamente al número de datos almacenados en la lista en cada momento

Desventajas de las listas enlazadas

- Acceso a posiciones intermedias con coste dependiente del tamaño de la lista
- Necesidad de memoria adicional para almacenar los objetos Node con sus atributos