Listas simplemente enlazadas

EII147-01-02 Introducción a las tecnologías de información
Escuela de Ingeniería Industrial
Pontificia Universidad Católica de Valparaíso
Rafael Mellado Silva
rafael.mellado@ucv.cl

Listas simplemente enlazadas

- Presentan un grado de flexibilidad mayor que los arreglos y matrices.
- Las acciones de insertar o eliminar un enésimo elemento no contemplan desplazamiento de los elementos restantes de la lista.
- Los elementos se almacenan en posiciones de memoria que no son continuas o adyacentes, por lo que cada elemento necesita almacenar la posición o dirección del siguiente elemento de la lista.

2

Listas simplemente enlazadas

- Son estructuras dinámicas: se asigna memoria para los elementos de la lista en la medida que es necesario.
- Cada elemento se almacena en una variable dinámica denominada nodo.
- En la lista simplemente enlazada, cada nodo apunta al nodo que contiene el elemento siguiente.

3

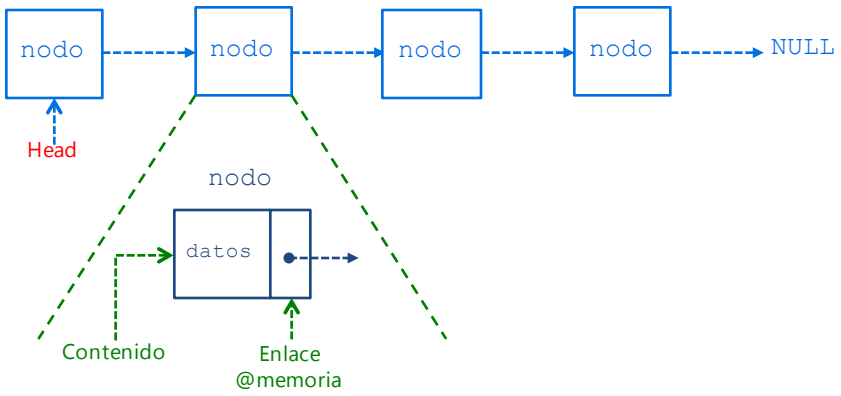
Listas simplemente enlazadas

- Los nodos de una lista contendrán datos del tipo declarado en la estructura del nodo. Por ejemplo:
  - Tipos primitivos (byte, int, boolean, char, etc.)
  - Referencias a objetos
- En los siguientes ejemplos consideraremos el uso de listas de enteros o Strings, aunque las técnicas que serán descritas son aplicables a cualquier otro "tipo" de lista.

4

Listas simplemente enlazadas

- A continuación se aprecia gráficamente una lista simplemente enlazada



5

Uso de listas

- Supongamos la clase **Presidente**.
- Los objetos de esta clase tienen un nombre y una referencia a un sucesor. El sucesor es un objeto de la misma clase:

6

Uso de listas

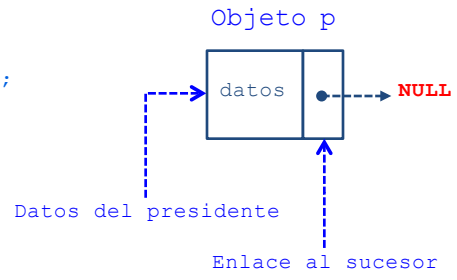
```
public class Presidente{
    private String Nombre;
    private Presidente sucesor;

    public Presidente(String n, Presidente s){
     nombre = n;
     sucesor = s;
    }
    public void setNombre(String n){
     nombre = n;
    }
    public String getNombre(){
     return nombre;
    }
    public void setSucesor(Presidente s){
     sucesor = s;
    }
    public Presidente getSucesor(){
     return sucesor;
    }
}
```

7

Uso de listas

- Suponga que una aplicación tiene una variable de tipo Presidente (llamada *p*) para la cual se crea un Presidente :

```
Presidente p = new Presidente("Alessandri", null);
```

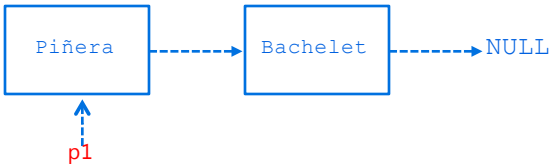


8

Uso de listas

- Suponga que una aplicación ahora tiene dos variables de tipo Presidente (llamadas p1 y p2), y ocurre lo siguiente:

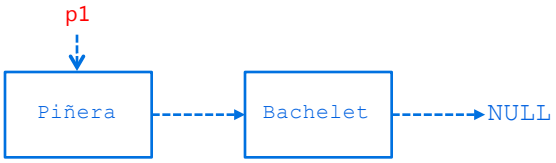
```
Presidente p2 = new Presidente("Bachelet" , null);
Presidente p1 = new Presidente("Piñera", p2);
```



9

Uso de listas

- Teniendo:



- Entonces, qué genera..?

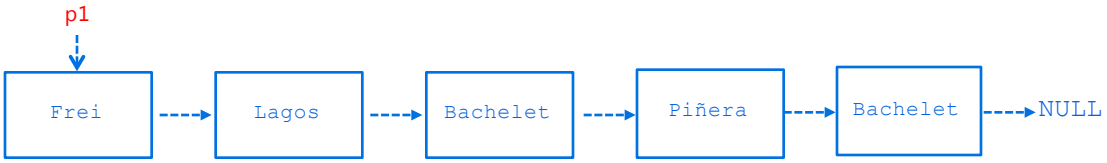
```
System.out.println( p1.getNombre() );
```

```
System.out.println( p1.getSucesor().getNombre() );
```

```
Presidente aux = p1.getSucesor();
System.out.println( aux.getNombre() );
```

10

Recorrido de listas




```
Presidente  aux  = p1;
while(  aux  != null  ){
     System.out.println(  aux.getNombre()  );
     aux  = aux.getSucesor();
}
```

11

Búsqueda de un elemento

- La búsqueda de un nombres se logra mediante el siguiente código:

```
String buscado = ... // aquí se asigna un valor
boolean está = false;
Presidente aux = p1;
while( aux != null ){
     if( aux.getNombre().equals( buscado ) )
          está = true;
     aux = aux.getSucesor();
}
if( está )
     System.out.println( buscado + " sí está" );
else
     System.out.println( buscado + " ní está" );
```

12

Implementación de listas

```
public class Lista{
   private Nodo head = null;
   public void agregarAlFinal(int data){
     ...
   }
   public void imprimirContenido(){
   ...
   }
   public boolean estaContenido(int data){
   ...
   }
   public boolean eliminar(int data){
   ...
   }
}
```

13

Implementación de listas

```
public class Nodo{
     private int data;
     private Nodo next;

     public Nodo(int d, Nodo n){
          data = d;
          next = n;
     }
     public int getData(){
          return data;
     }
     public Nodo getNext(){
          return next;
     }
     public void setNext(Nodo n){
          next = n;
     }
}
```

14

Insertar elementos

```
public class Lista{
   ...
   public void agregarAlFinal(int dato){
     Nodo nuevo = new Nodo(dato, null);
     if( head == null )
          head = nuevo;
     else{
          Nodo aux = head;
          while( aux.getNext() != null)
               aux = aux.getNext();
          aux.setNext( nuevo );
     }
    }
   ...
}
```

- ¿Cómo sería para agregar al principio u ordenado?

15

Mostrar contenido

```
public class Lista{
  ...
public void imprimirContenido(){
   Nodo aux = head;
   while( aux != null ){
        System.out.print( aux.getData() + "; " );
        aux = aux.getNext();
   }
   System.out.println();
    ...
}
```

16

Búsqueda de contenido

```
public class Lista{
  ...
  public boolean estaContenido(int data){
     Nodo aux = head;
     while( aux != null ){
        if( data == aux.getData() )
          return true;
        aux = aux.getNext();
     }
     return false;
  }
  ...
}
```

17

Eliminar un elemento

```
public class Lista{
  ...
  public boolean eliminar(int data){
     if( head != null)
      if( head.getData() == data ){
          head = head.getNext();
          return true;
           }else{
               Nodo aux = head;
          while( aux.getNext() != null ){
             if( aux.getNext().getData() == data ){
               aux.setNext( aux.getNext().getNext() );
               return true;
             }
             aux = aux.getNext();
              }
       }
     return false;
  }
     ...
}
```

18

Ejercicios

- Ejercicio 1: Implementar la clase Lista (y la clase Nodo).
- Ejercicio 2: Utilizar la clase Lista para mantener datos en una aplicación.
- Ejercicio 3: Visualizar cómo la clase Lista puede ser utilizada el interior de una clase Estantería, para almacenar objetos de tipo Producto (en lugar de utilizar un arreglo).

19

Listas simplemente enlazadas

EII147-01-02 Introducción a las tecnologías de información
Escuela de Ingeniería Industrial
Pontificia Universidad Católica de Valparaíso
Rafael Mellado Silva
rafael.mellado@ucv.cl

Especificación sobre listas

EII147-01-02 Introducción a las tecnologías de información
Escuela de Ingeniería Industrial
Pontificia Universidad Católica de Valparaíso
Rafael Mellado Silva
rafael.mellado@ucv.cl

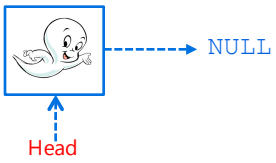
Nodo fantasma

- En el esquema propuesto se deben hacer excepciones al insertar y eliminar el nodo del comienzo de la lista.
- El manejo se simplifica si se utiliza un nodo "fantasma":
  - Es un nodo siempre presente en la lista
  - Su contenido es irrelevante (el valor u objeto contenido no forma parte de la lista).

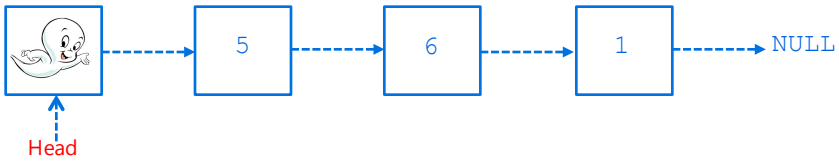
2

Nodo fantasma

- Lista vacía:



- Lista con elementos:



3

Nodo fantasma

```
public class Lista{
  Nodo head;
  public Lista(){
    head = new Nodo(0, null);
  }
  ...
}
```

Valor irrelevante

4

Nodo fantasma: insertar

```
public void agregarAlFinal(int dato){
    Nodo aux = head;
    while( aux.getNext() != null)
        aux = aux.getNext();
    aux.setNext( new Nodo(dato, null) );
    }
    }
```

5

Nodo fantasma: eliminar

```
public boolean eliminar(int data){
      Nodo aux = head;
         while( aux.getNext()  != null ){
          if( aux.getNext().getData()  == data ){
            aux.setNext(  aux.getNext().getNext()  );
            return true;
          }
          aux = aux.getNext();
         }
       return false;
}
```

6

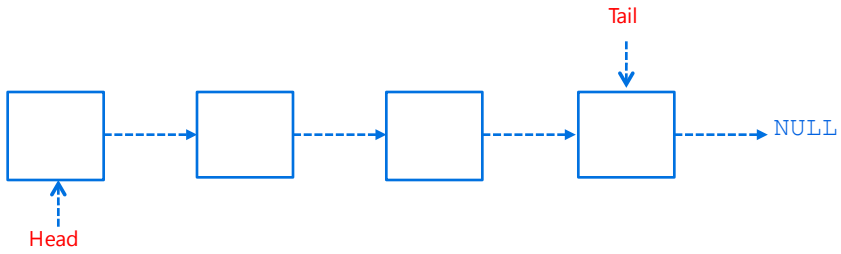
Nodo fantasma: eliminar

- Los recorridos descritos anteriormente requieren que todas las veces sea encontrado el último elemento de la lista.
- Más conveniente: tener una variable de instancia que siempre referencie al último elemento de la lista.
- Esto aplica a listas con o sin nodo fantasma (con pequeños cambios).

7

Tail

- La variable de instancia **`tail`** mantiene siempre la referencia al último elemento:



8

Tail

- La variable de instancia **tail** mantiene siempre la referencia al último elemento:

```
public class Lista{
  Nodo head, tail;
  public Lista(){
    head = new Nodo(0, null);
    tail = head;
  }
  ...
}
```

9

Tail

- El método **agregarAlFinal** ya no requiere recorrer la lista para ubicar el último nodo:

```
public void agregarAlFinal(int dato){
     Nodo aux = new Nodo(dato, null) ;
     tail.setNext( aux );
     tail = aux;
}
```

Versión con nodo fantasma

- La varaible **tail** es actualizada después de la inserción.
- Notar que el procedimiento de eliminación debe actualizar la referencia tail si se remueve el último nodo de la lista.

10

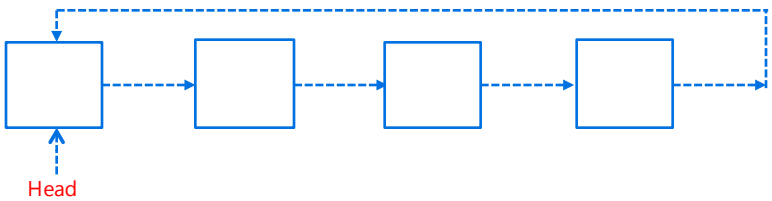
Resumen sobre listas

- Útiles para guardar un número no predefinido de elementos.
- Distintas disciplinas para mantener los datos ordenados (y para removerlos).
- El acceso a los nodos es secuencial; el recorrido es en una sola dirección (Ejercicio: confrontar con arreglos)
- Están diseñadas para un acceso fácil al nodo siguiente y al anterior.
- Cada nodo contiene dos referencias: una apuntando al nodo siguiente, y otra apuntando al nodo anterior.

11

Resumen sobre listas

- El acceso a los nodos sigue siendo secuencial.
- La técnica del nodo fantasma puede ser útil también en este tipo de lista.
- Estar abierto a definir y utilizar otras estructuras.
- Ejemplo: Lista simplemente enlazada y circular:



12

Especificación sobre listas

EII147-01-02 Introducción a las tecnologías de información
Escuela de Ingeniería Industrial
Pontificia Universidad Católica de Valparaíso
Rafael Mellado Silva
rafael.mellado@ucv.cl