

# Sesión 22: Cadena *simplemente* enlazada

## Programación 2

---

Ángel Herranz

Abril 2019

Universidad Politécnica de Madrid

# En capítulos anteriores

- 👍 Tema 1: Clases y Objetos
- 👍 Tema 2: Colecciones acotadas de Objetos
- 👍 Tema 4: Tipos Abstractos de Datos
- 👍 Tema 3: Programación Modular
- 👍 Tema 5: Herencia y Polimorfismo
- 👍 Tema 6: Excepciones
- 🕒 Tema 7: Implementación de TADs lineales
  - *A long time ago...* `Nodo<T>`
  - *Arrays* redimensionables

# En el capítulo de hoy

Cadenas

enlazadas

# En el capítulo de hoy

## Cadenas *simplemente* enlazadas

# En el capítulo de hoy

## Cadenas *simplemente* enlazadas

¿Por qué es importante?

# En el capítulo de hoy

## Cadenas *simplemente* enlazadas

¿Por qué es importante?

- ⚠ Base de la implementación de otras estructuras
- ⚠ Magnífico ejercicio de algoritmia
- ⚠ Manejo de punteros

# Listas

- Hoy: cadenas enlazadas para representar *listas*
- Pero... ¿*qué son las listas?*

# Listas

- Hoy: cadenas enlazadas para representar *listas*
- Pero... ¿*qué son las listas?*

Colección



# Listas

- Hoy: cadenas enlazadas para representar *listas*
- Pero... ¿*qué son las listas?*

Colección ordenada

# Listas

- Hoy: cadenas enlazadas para representar *listas*
- Pero... ¿*qué son las listas?*

Colección ordenada de datos que pueden repetirse

# Listas

- Hoy: cadenas enlazadas para representar *listas*
- Pero... ¿*qué son las listas?*

Colección ordenada de datos que pueden repetirse, potencialmente vacía

# Listas

- Hoy: cadenas enlazadas para representar *listas*
- Pero... ¿qué son las listas?

Colección ordenada de datos que pueden repetirse, potencialmente vacía

- Pongamos en común lo que vemos

Lista vacía            [ ]

Un dato              [ 42     ]

Varios datos        [ 21,    13, 8, 5, 3, 2, 1 ]

Incluso repetidos [ 10,    0, 0, 2, 7, 7 ]

# Listas

- Hoy: cadenas enlazadas para representar *listas*
- Pero... ¿qué son las listas?

Colección ordenada de datos que pueden repetirse, potencialmente vacía

- Pongamos en común lo que vemos

Lista vacía [ ]

Un dato [ 42 ]

Varios datos [ 21, 13, 8, 5, 3, 2, 1 ]

Incluso repetidos [ 10, 0, 0, 2, 7, 7 ]

*primero*    *resto*

# Listas

- Hoy: cadenas enlazadas para representar *listas*
- Pero... ¿qué son las listas?

Colección ordenada de datos que pueden repetirse, potencialmente vacía

- Pongamos en común lo que vemos

Lista vacía [ ]

Un dato [ 42 ]

Varios datos [ 21, 13, 8, 5, 3, 2, 1 ]

Incluso repetidos [ 10, 0, 0, 2, 7, 7 ]

*primero*    *resto*  
*cabeza*    *cola*

# Dos casos

Una lista...

# Dos casos

Una lista...

1. puede ser vacía



Una lista...

1. puede ser vacía
2. o puede tener un primero y un resto

## Una lista...

1. puede ser vacía
2. o puede tener un primero y un resto donde el primero es un dato y

## Una lista...

1. puede ser vacía
2. o puede tener un primero y un resto donde el primero es un dato y el resto es una lista

# Dos casos (definición recursiva)

Una **lista**...

1. puede ser vacía
2. o puede tener un primero y un resto donde el primero es un dato y el **resto es una lista**

# ¿¡Qué es esto!?

 2' Digerir

```
public class Nodo<T> {  
    public T dato;  
    public Nodo<T> siguiente;  
  
    public Nodo(T dato) {  
        this.dato = dato;  
        siguiente = null;  
    }  
}
```

# ¿Qué es esto!?

```
public class Nodo<T> {  
    public T dato;  
    public Nodo<T> siguiente;
```

```
    public Nodo(T dato) {  
        this.dato = dato;  
        siguiente = null;  
    }  
}
```

🕒 2' Digerir

Declaración

```
Nodo<Integer> lista;
```

# ¿¡Qué es esto!?

```
public class Nodo<T> {  
    public T dato;  
    public Nodo<T> siguiente;  
  
    public Nodo(T dato) {  
        this.dato = dato;  
        siguiente = null;  
    }  
}
```

🕒 2' Digerir

Nodo<Integer> lista;

lista =

new Nodo<Integer>(21);

Declaración

Construcción

# ¿¡Qué es esto!?

```
public class Nodo<T> {  
    public T dato;  
    public Nodo<T> siguiente;  
  
    public Nodo(T dato) {  
        this.dato = dato;  
        siguiente = null;  
    }  
}
```

🕒 2' Digerir

```
Nodo<Integer> lista;  
  
lista =  
    new Nodo<Integer>(21);  
  
System.out.println(  
    lista.dato  
);
```

Declaración

Construcción

Acceso



# ¿¡Qué es esto!?

```
public class Nodo<T> {  
    public T dato;  
    public Nodo<T> siguiente;  
  
    public Nodo(T dato) {  
        this.dato = dato;  
        siguiente = null;  
    }  
}
```

🕒 2' Digerir

Declaración

Nodo<Integer> lista;

Construcción

lista =  
 new Nodo<Integer>(21);

Acceso

System.out.println(  
 lista.dato  
);

Modificación

lista.resto =  
 new Nodo<Integer>(13);

# ¿¡Qué es esto!?

```
public class Nodo<T> {  
    public T dato;  
    public Nodo<T> siguiente;  
  
    public Nodo(T dato) {  
        this.dato = dato;  
        siguiente = null;  
    }  
}
```

¡Dibujar!

⌚ 2' Digerir

```
Nodo<Integer> lista;  
  
lista =  
    new Nodo<Integer>(21);  
  
System.out.println(  
    lista.dato  
);  
  
lista.resto =  
    new Nodo<Integer>(13);
```

Declaración

Construcción

Acceso

Modificación

# El poder de `null`

Usaremos **`null`** para representar [ ]

# Código con cadenas enlazadas

- Durante el resto de la clase vamos a utilizar
- `l` para referirnos a una cadena enlazada de strings (i.e. una variable de tipo `Nodo<String>`)
- `i` para referirnos a una variable **`int`**
- `s` para referirnos a una variable `String` (no **`null`**)



`l` puede contener **`null`** o una referencia

# Código con cadenas enlazadas

- Durante el resto de la clase vamos a utilizar
- `l` para referirnos a una cadena enlazada de strings (i.e. una variable de tipo `Nodo<String>`)
- `i` para referirnos a una variable **`int`**
- `s` para referirnos a una variable `String` (no **`null`**)



`l` puede contener **`null`** o una referencia

- `n` para referirnos a una variable **`int`**
- `encontrado` para referirnos a una variable **`boolean`**

# Crear una lista vacía

# Crear una lista vacía

```
l = null;
```

# Comprobar que `l` es una lista vacía



# Comprobar que `l` es una lista vacía

```
l == null
```

# Insertar s al principio de l

# Insertar s al principio de l

```
Nodo<String> resto;  
resto = l;  
l = new Nodo<String>(s);  
l.siguiente = resto;
```

# Insertar s al principio de l

```
Nodo<String> resto;  
resto = l;  
l = new Nodo<String>(s);  
l.siguiente = resto;
```

¿Y si l está vacía?



# Prepara un main

```
public class OperacionesCadena {  
    public static void main(String[] args) {  
        Nodo<String> l;  
        String s;  
        int i;  
        int n;  
        boolean encontrado;  
  
        l = null;  
  
        if (l == null) System.out.println("Vacía");  
        else System.out.println("No vacía");  
  
        Nodo<String> resto;  
        resto = l;  
        l = new Nodo<String>(s);  
        l.siguiente = resto;  
  
        // Más operaciones aquí  
    }  
}
```

# Calcular el número de datos en $l^1$

---

<sup>1</sup>y dejarlo en  $n$



# Imprimir todos los datos en 1

# Insertar s al final de l



 Insertar  $s$  en la posición  $i$  de  $l$

# Comprobar si $s$ está en $\mathcal{L}^2$

---

<sup>2</sup>usar la variable encontrado

# Borrar el primero de 1

# Borrar el último de 1

 Borrar el dato en la posición  $i$  de  $\mathbf{l}$

# Borrar el dato s de l

# Insertar s en orden en l