Sesión 14: Cadena *simplemente* enlazada (1/2)

Programación 2

Ángel Herranz

2020-2021

Universidad Politécnica de Madrid

En capítulos anteriores

- 👍 Tema 1: Intro a POO
- 👍 Tema 2: Clases y Objetos y TADs y módulos
- 💻 Ejercicio práctico el día 21 de abril, a entregar en Moodle

En el capítulo de hoy

Tema 3: Cadenas enlazadas

En el capítulo de hoy

Tema 3: Cadenas *simplemente* enlazadas

En el capítulo de hoy

Tema 3: Cadenas *simplemente* enlazadas

¿Por qué es importante?

En el capítulo de hoy

Tema 3: Cadenas *simplemente* enlazadas

¿Por qué es importante?

- Fundamento de estructuras de datos *ilimitadas*
- 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

Dos casos

Una lista...

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

Dos casos

Una lista...

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

Dos casos

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

```
class NodoInt {
  int dato;
  NodoInt siguiente;
}
```

¿¡Qué es esto!?

```
class NodoInt {
  int dato;
  NodoInt siguiente;
}
```

2' Digerir

Declaración

```
NodoInt lista;
```

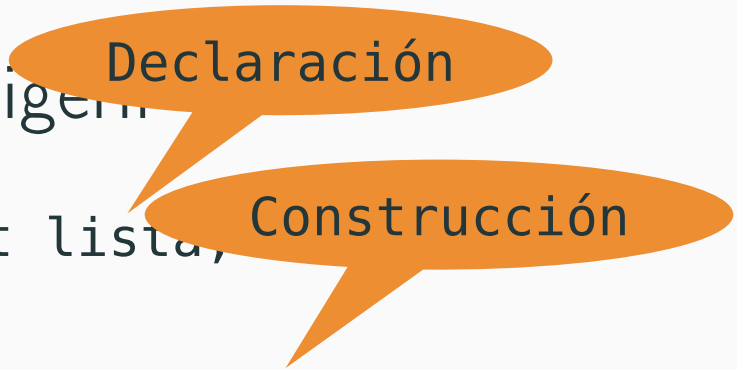
¿¡Qué es esto!?

```
class NodoInt {
  int dato;
  NodoInt siguiente;
}
```

2' Digen

```
NodoInt lista;

lista = new NodoInt();
```



¿¡Qué es esto!?

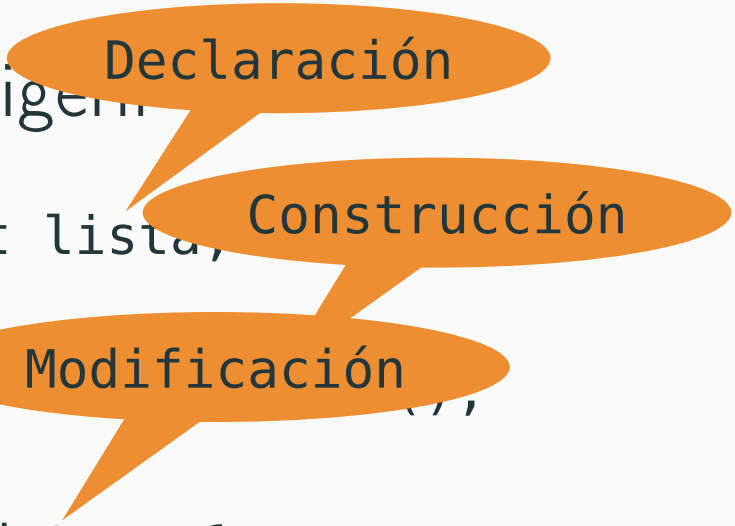
```
class NodoInt {
  int dato;
  NodoInt siguiente;
}
```

2' Digen

```
NodoInt lista;

lista

lista.dato = 1;
```



¿¡Qué es esto!?

```
class NodoInt {
  int dato;
  NodoInt siguiente;
}
```

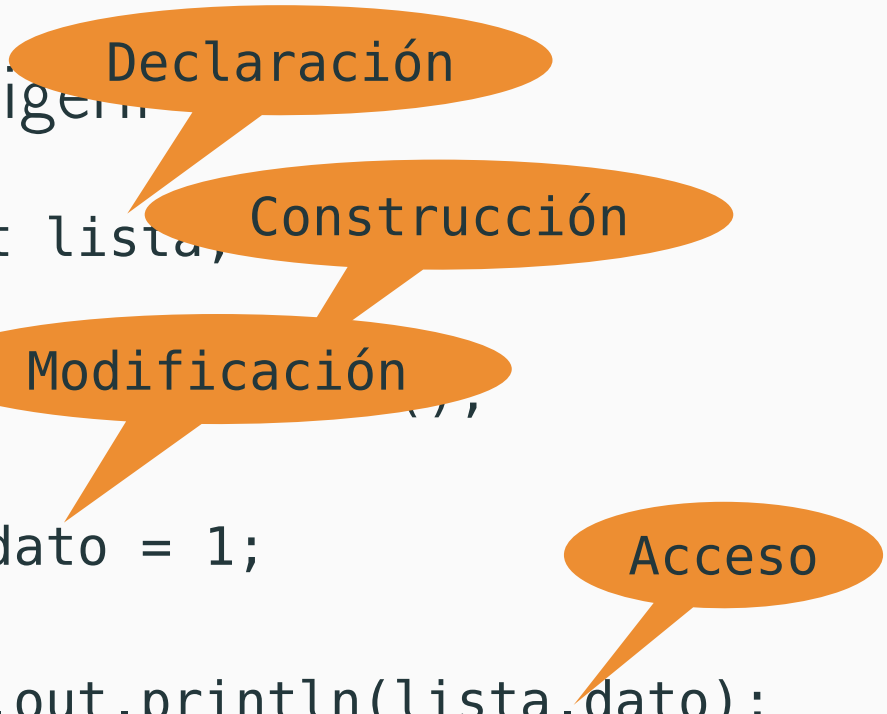
2' Digen

```
NodoInt lista;

lista                    ();

lista.dato = 1;

System.out.println(lista.dato);
```



¿¡Qué es esto!?

```
class NodoInt {
  int dato;
  NodoInt siguiente;
}
```

¡Dibujad listas y pensad en como construirlas!

2' Digen

```
NodoInt lista;

lista

lista.dato = 1;

System.ou                    a.dato);

lista.siguiente = new NodoInt();
```

Declaración

Construcción

Modificación

Acceso

Modificación

¿¡Qué es esto!?

```
class NodoInt {
  int dato;
  NodoInt siguiente;
}
```

¡Dibujad listas y pensad en como construirlas!

2' Digerir

```
NodoInt lista;

lista = new NodoInt();

lista.dato = 1;

System.out.println(lista.dato);

lista.siguiente = new NodoInt();

lista.siguiente.dato = 2;
```

Declaración

Construcción

Modificación

Acceso

Modificación

Modificación

El poder de `null`

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

¿Y si quiero una cadena de *strings*?

Por ahora: *copy & paste, find & replace*

```
class NodoInt {
  int dato;
  NodoInt siguiente;
}
```

```
class NodoStr {
  String dato;
  NodoStr siguiente;
}
```

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 `NodoStr`)
- `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 `NodoStr`)
- `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 en `l`

Crear una lista vacía en `l`

```
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`

```
NodoStr resto;
resto = l;
l = new NodoStr();
l.dato = s;
l.siguiente = resto;
```

Insertar `s` al principio de `l`

```
NodoStr resto;
resto = l;
l = new NodoStr();
l.dato = s;
l.siguiente = resto;
```

¿Y si `l` está vacía?

Prepara un `main` y prueba

```
public class OperacionesCadena {
  public static void main(String[] args) {
    NodoStr l;
    String s;
    int i;
    int n;
    boolean encontrado;

    // Crear lista vacía
    {
      l = null;
    }

    // Comprobar que l es vacía
    {
      if (l == null) System.out.println("Vacía");
      else System.out.println("No vacía");
    }

    ...
```

Prepara un `main` y prueba

```
public class OperacionesCadena {

  private static int longitud(NodoStr l) {
    int longitud = 0;
    while (l != null) {
      longitud++;
      l = l.siguiente;
    }
    return longitud;
  }

  public static void main(String[] args) {

    ...
    System.out.prinln(longitud(l));
    ...

  }
}
```

Puedes crear funciones y usarlas

Más operaciones i

- Devolver en `s` el primero de `l`
- Devolver en `n` el número de datos en `l`
- Imprimir todos los datos en `l`
- Insertar `s` al final de `l`
- Devolver en `s` el último de `l`
- Borrar el primero de `l`
- Borrar el último de `l`

Más operaciones ii

- Devolver en `s` el `i`-ésimo de `l`
- Cambiar la posición `i` de `l` por `s`
- Insertar `s` en la posición `i` de `l`
- Devolver en `encontrado` si `s` está en `l`
- Borrar el dato en la posición `i` de `l`
- Borrar el dato `s` de `l`
- Insertar `s` en orden en `l`