Representación de estructuras de datos con nodos enlazados

Estructuras de Datos

Grado en Ingeniería Informática, del Software y Computadores

Universidad de Málaga

Representación de estructuras en Java

En Java la representación de una estructura de datos puede estar basada en:

- Arrays (`ArrayList`, `ArrayStack`, etc.)
- Nodos enlazados (`LinkedList`, `LinkedStack`, etc.)

Los arrays se estudiaron a fondo en primero

Este curso estudiaremos a fondo los **nodos enlazados**

Secuencia de nodos en Haskell

El tipo `Seq a` permite representar una secuencia de nodos:

```
Data Seq a = Empty
           | Node a (Seq a)
```

Algunos valores posibles:

```
Empty
Node “hola” Empty
Node 7 (Node 3 (Node 5 (Node 9 Empty)))
```

Un constructor **Node** por cada elemento de la secuencia

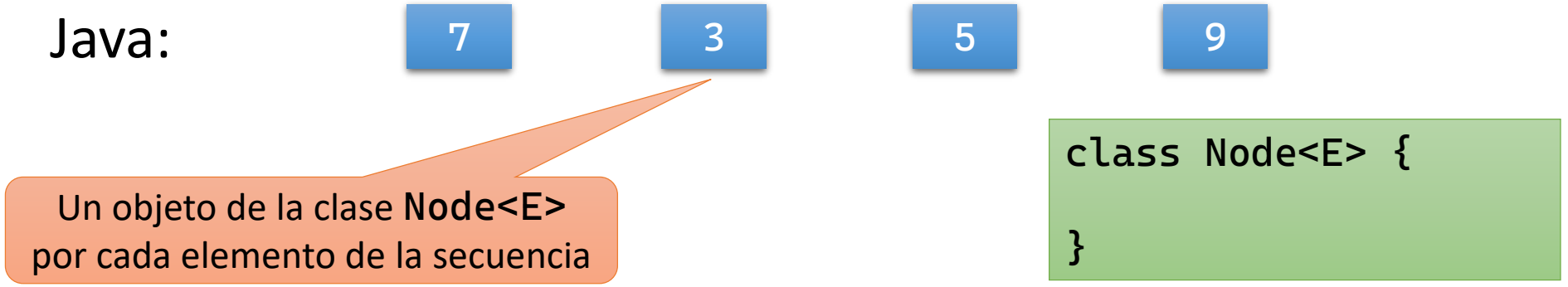
De Haskell a Java (I)

En Java queremos hacer algo similar:

- Los nodos deben ser **objetos** de una clase genérica `Node<E>`
- Debe haber un **objeto** (`Node<E>`) por cada elemento de la secuencia

Haskell: `Node 7 (Node 3 (Node 5 (Node 9 Empty)))`

Java:



De Haskell a Java (II)

Tenemos que definir la clase genérica `Node<E>` en Java:

- ¿Qué atributos debe tener?

Haskell: `Node 7 (Node 3 (Node 5 (Node 9 Empty)))`

Java:

Un objeto de la clase `Node<E>` por cada elemento de la secuencia

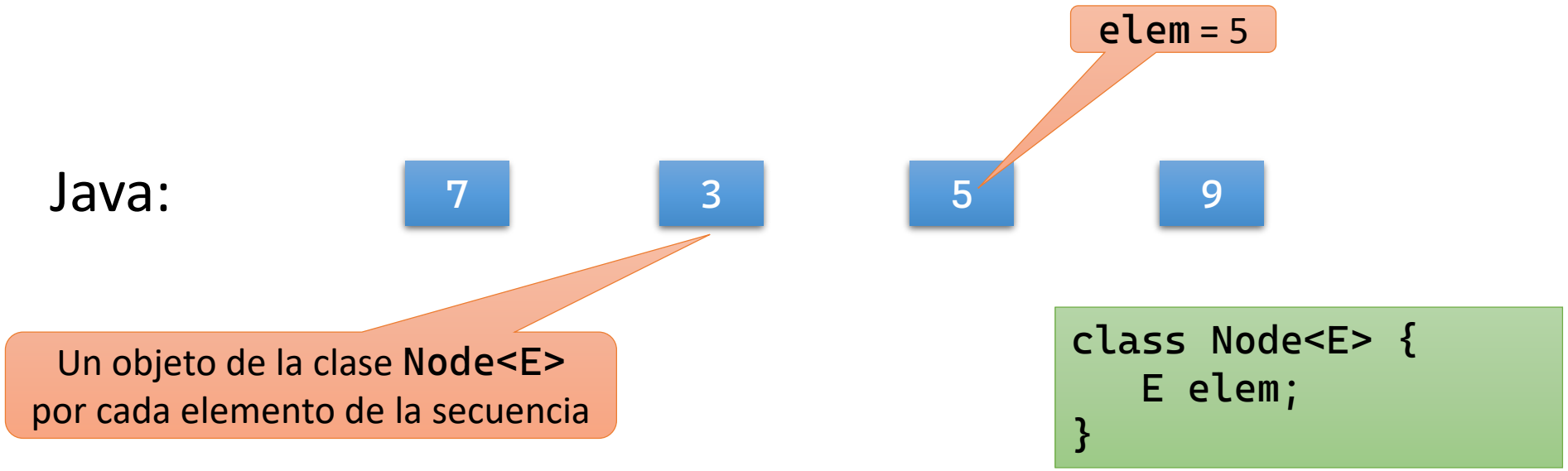
```
class Node<E> {
   // ¿atributos?
}
```

De Haskell a Java (III)

Un atributo `elem` de tipo `E` para almacenar cada elemento

Haskell: `Node 7 (Node 3 (Node 5 (Node 9 Empty)))`

Java:



De Haskell a Java (IV)

¿Qué falta?

Haskell: `Node 7 (Node 3 (Node 5 (Node 9 Empty)))`

Java:

Un objeto de la clase `Node<E>` por cada elemento de la secuencia

```
class Node<E> {
   E elem;
}
```

De Haskell a Java (V)

Falta decir que al 7 le sigue el 3, al 3 el 5, y al 5 el 9…

- Tenemos 4 objetos (nodos) no relacionados entre ellos

Haskell: `Node 7 (Node 3 (Node 5 (Node 9 Empty)))`

Java:

Un objeto de la clase `Node<E>` por cada elemento de la secuencia

```
class Node<E> {
   E elem;
}
```

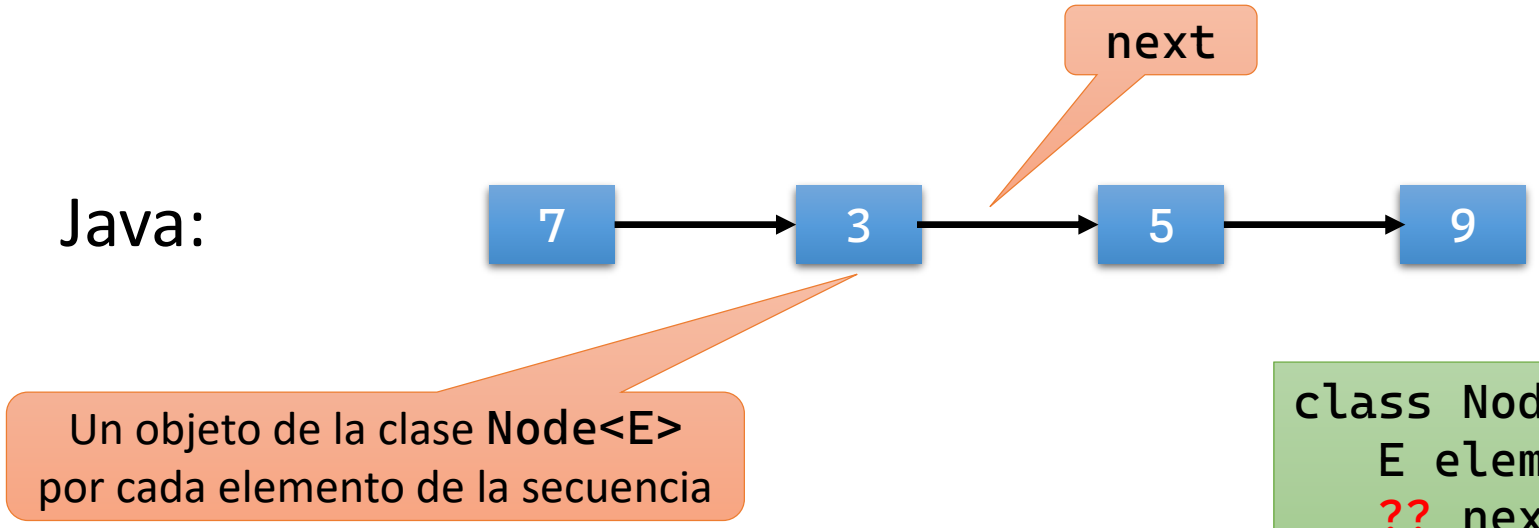
De Haskell a Java (VI)

Los nodos deben estar **enlazados**; cada nodo apunta al siguiente:

- Usamos un atributo `next` para almacenar el enlace

Haskell: `Node 7 (Node 3 (Node 5 (Node 9 Empty)))`

Java:



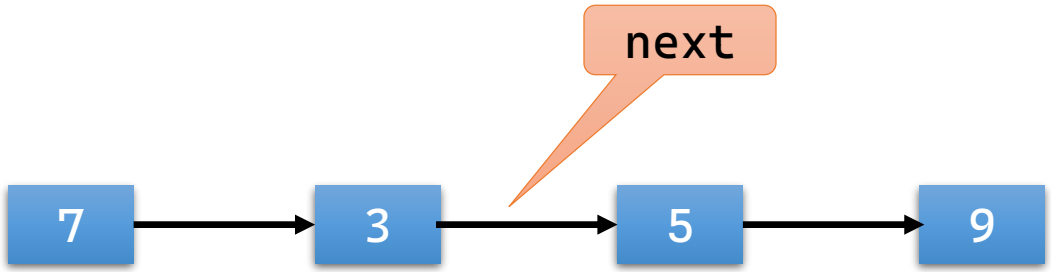

```
class Node<E> {
   E elem;
   ?? next;
}
```

De Haskell a Java (VII)

¿De qué tipo es el atributo `next`?

Haskell: `Node 7 (Node 3 (Node 5 (Node 9 Empty)))`

Java:



Un objeto de la clase `Node<E>` por cada elemento de la secuencia

```
class Node<E> {
   E elem;
   ?? next;
}
```

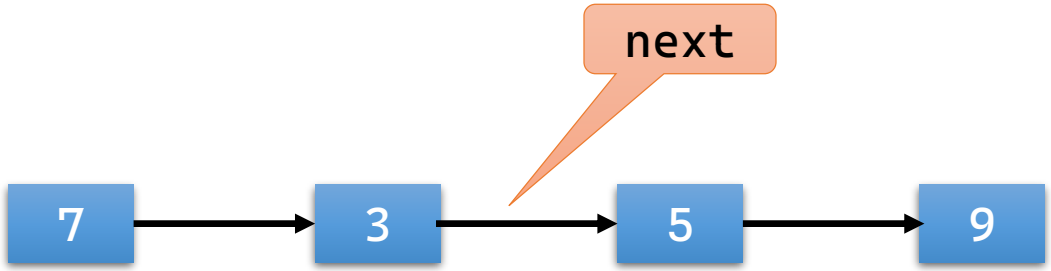
De Haskell a Java (VIII)

¿De qué tipo es el atributo `next`?

- `next` es una referencia al siguiente nodo: su tipo es `Node<E>`

Haskell: `Node 7 (Node 3 (Node 5 (Node 9 Empty)))`

Java:



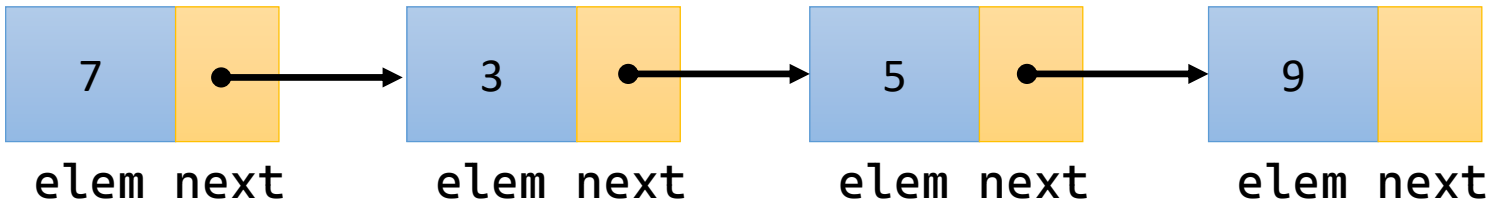
Un objeto de la clase `Node<E>` por cada elemento de la secuencia

```
class Node<E> {
   E elem;
   Node<E> next;
}
```

Secuencia de nodos enlazados en Java

Observa que la clase **Node<E>** es **recursiva**

```
class Node<E> {
    E elem;
    Node<E> next;
}
```

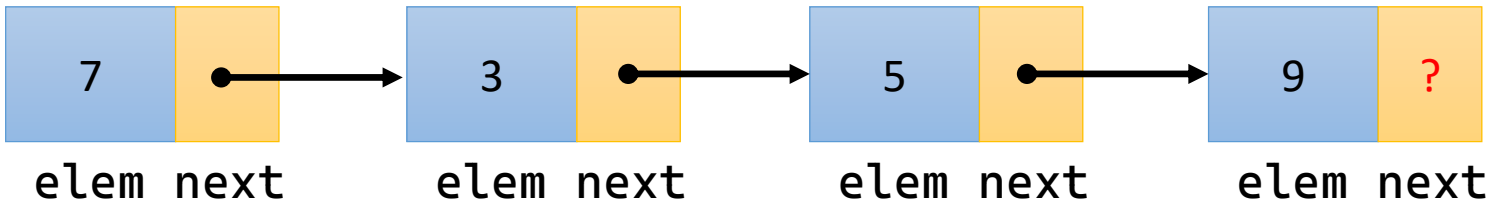


¿Qué hacemos con `Empty`?

Al constructor Haskell `Node` le corresponde la clase Java `Node<E>`
¿Qué ocurre con `Empty`? ¿Cómo señalamos el final de la secuencia?

Haskell: `Node 7 (Node 3 (Node 5 (Node 9 Empty)))`

```
class Node<E> {
   E elem;
   Node<E> next;
}
```

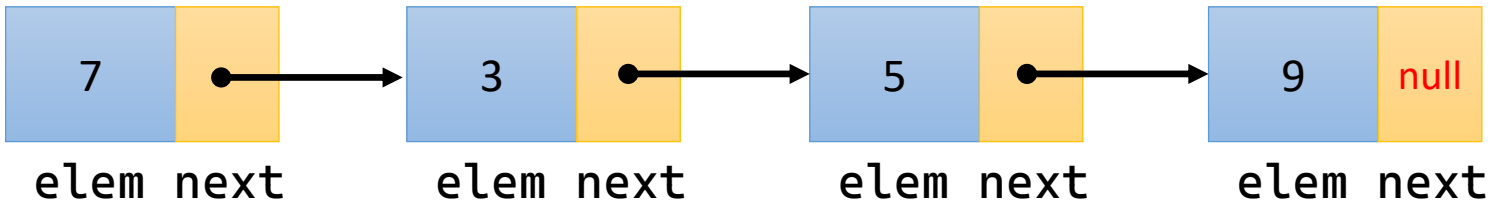


`Empty` es `null`

El último nodo no tiene siguiente
Su atributo `next` vale `null`

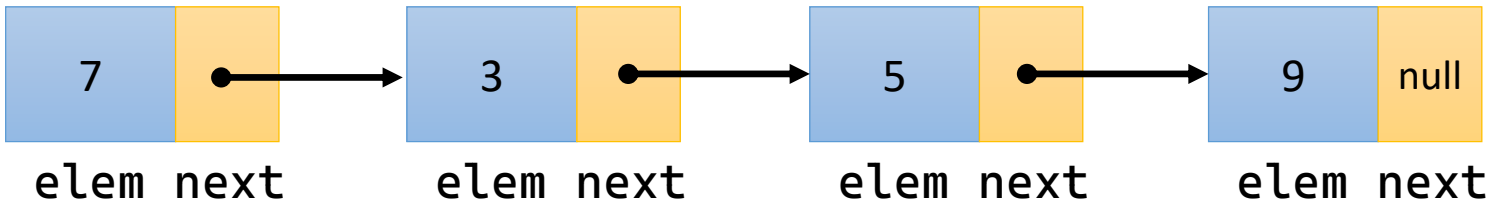
Haskell: `Node 7 (Node 3 (Node 5 (Node 9 Empty)))`

```
class Node<E> {
   E elem;
   Node<E> next;
}
```



¿Qué falta para terminar?

```
class Node<E> {
   E elem;
   Node<E> next;
}
```

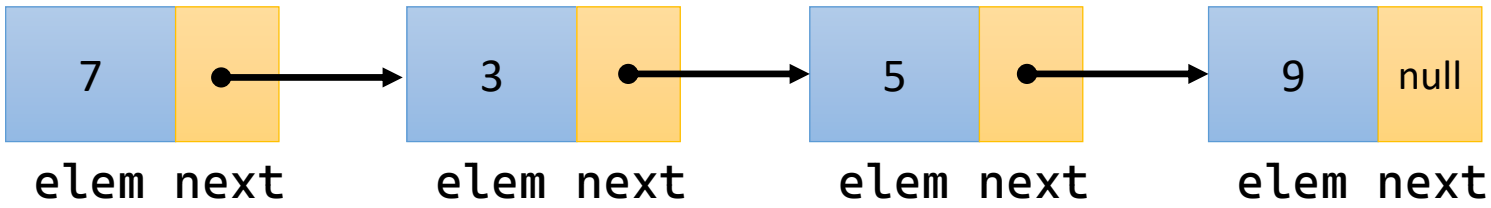


No tenemos acceso a la secuencia

Tenemos todos los nodos enlazados, pero:

- No tenemos acceso a la secuencia
- Necesitamos una referencia al **primer** nodo de la secuencia

```
class Node<E> {
   E elem;
   Node<E> next;
}
```



De Haskell a Java Clásico

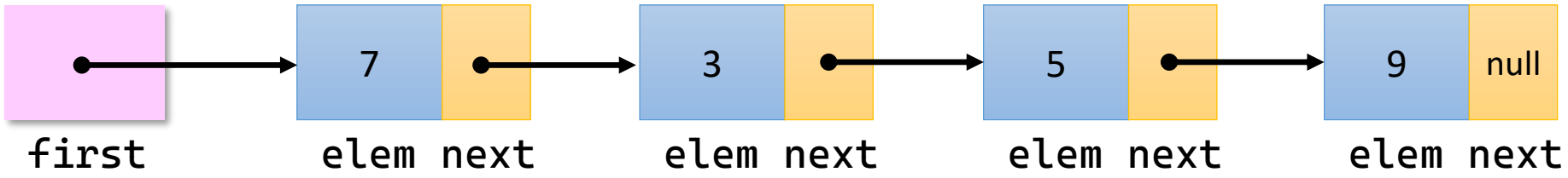
```
Data Seq a = Empty
           | Node a (Seq a)
```

```
public class Seq<T> {

   private static class Node<E> {
      E elem;
      Node<E> next;
   }

   private Node<T> first;
}
```

Node<E> es una clase anidada estática de **Seq<T>**



De Haskell a Java Moderno

```
Data Seq a = Empty
           | Node a (Seq a)




length :: Seq a -> Int
length Empty = 0
Length (Node _ s) = 1 + length s
```

```
public sealed interface Seq<T> {
   record Empty<T>() implements Seq<T> {}
   record Node<T>(T elem, Seq<T> next) implements Seq<T> {}
}


public static <T> int length(Seq<T> seq) {
  return switch(seq) {
     case Empty<T>() -> 0;
     case Node<T>(T x, Seq<T> s) -> 1 + length(s);
   };
}
```

tipo algebraico (inmutable)

recursividad

patrones

recursividad

Más información: [Data Oriented Programming in Java, Goetz & Bryant]