ESTRUCTURA DE DATOS

Tecnológico Nacional de Culiacán
Instituto Tecnológico de Culiacán
Ingeniería de Sistemas

Mtra. María Lourdes Armenta Lindoro

Culiacán, Sinaloa. AGOSTO de 2023

TEMARIO

Unidad III. Estructuras Lineales

- 3.1 Pilas. 3.1.1 Representación en memoria. 3.1.2 Operaciones básicas. 3.1.3 Aplicaciones.
- 3.2 Colas. 3.2.1 Representación en memoria. 3.2.2 Operaciones básicas. 3.2.3 Tipos de colas: simples, circulares y bicolas. 3.2.4 Aplicaciones.
- 3.3 Listas. 3.3.1 Operaciones básicas. 3.3.2 Tipos de listas: simplemente enlazadas, doblemente enlazadas y circulares. 3.3.3 Aplicaciones

En programación, “estructura” equivale a “colección” o “conjunto”

Por lo que una estructura de datos representa una colección de datos

Las estructuras contienen datos que pueden ser coherentes u homogéneos.

Ejemplo: Atributos de una clase Java componen un conjunto de datos que son coherentes.

```
public class Persona{
    private String dni;
    private String nombre;
    private Fecha FechaNacimiento;
    // Métodos get y set para los atributos.
}
```

```
public class Fecha{
    private int dia;
    private int mes;
    private int anio;
    //Métodos get y set para los atributos.
}
```

ESTRUCTURAS ESTÁTICAS

- Cuando la cantidad de elementos que contiene es fija y no puede ser modificada durante le ejecución de un programa.
- Por ejemplo arrays y clases
- La cantidad de elementos de una clase podrá contener dependerá de la cantidad de atributos que se declaran cuando se programa
- En el caso de los array, aunque en Java se declara su capacidad física en tiempo de ejecución, una vez declarada esta capacidad, el array permanecerá estático y no podrá ser redimensionado.

ESTRUCTURAS DINÁMICAS

- Una estructura dinámica consiste en una colección de datos cuya capacidad física puede ser modificada Durante la ejecución de un programa.
- Si se necesitan más capacidad porque siquiera contener una mayor cantidad de elementos entonces estructura dinámica podrá crecer. Y si ya no se necesita tanta capacidad porque decidimos desprender de alguno de los elementos que contenía la colección, la estructura dinámica podrá decrecer y, de este modo, insumo una menor cantidad de amor.
- Cuando los elementos que contiene una estructura dinámica están almacenados uno a continuación del otro, decimos que se trata de una estructura línea.
- Por el contrario si para un determinado elemento tengo una colección

TIPOS DE ESTRUCTURAS LINEALES

- Listas
- Pilas
- Colas
- Tablas de Dispersión (Tablas Hash)

- Estructura Lineal

Cada componente tiene un único sucesor y un único predecesor con excepción del último y el primero.

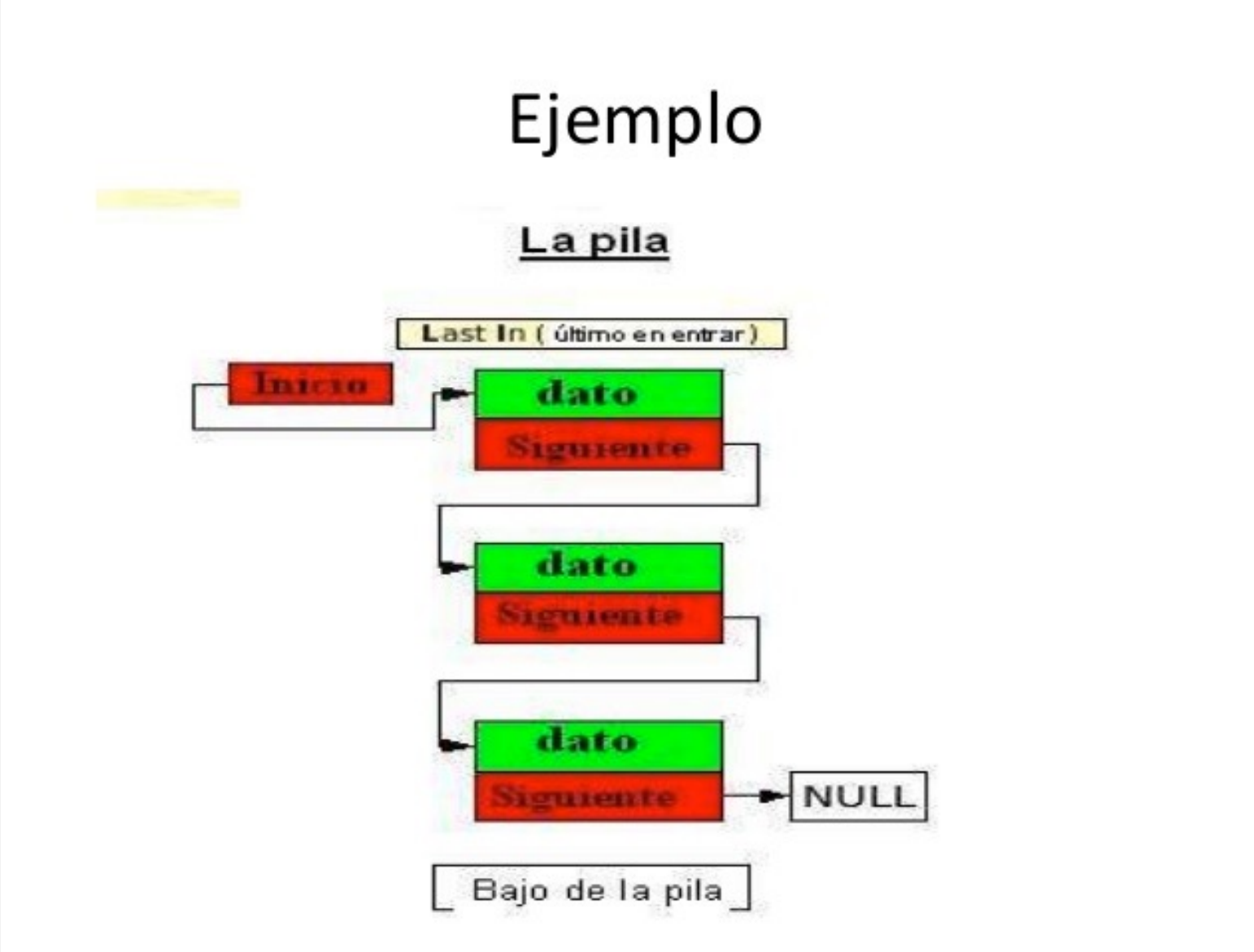
- Estructura No líneal

Cada componente puede tener varios sucesores y varios predecesores.

OPERACIONES BÁSICAS PARA ESTRUCTURAS LINEALES

- **crear** la secuencia vacía
- **añadir** un elemento a la secuencia
- **borrar** un elemento a la secuencia
- **consultar** un elemento de la secuencia
- comprobar si la secuencia está **vacía**

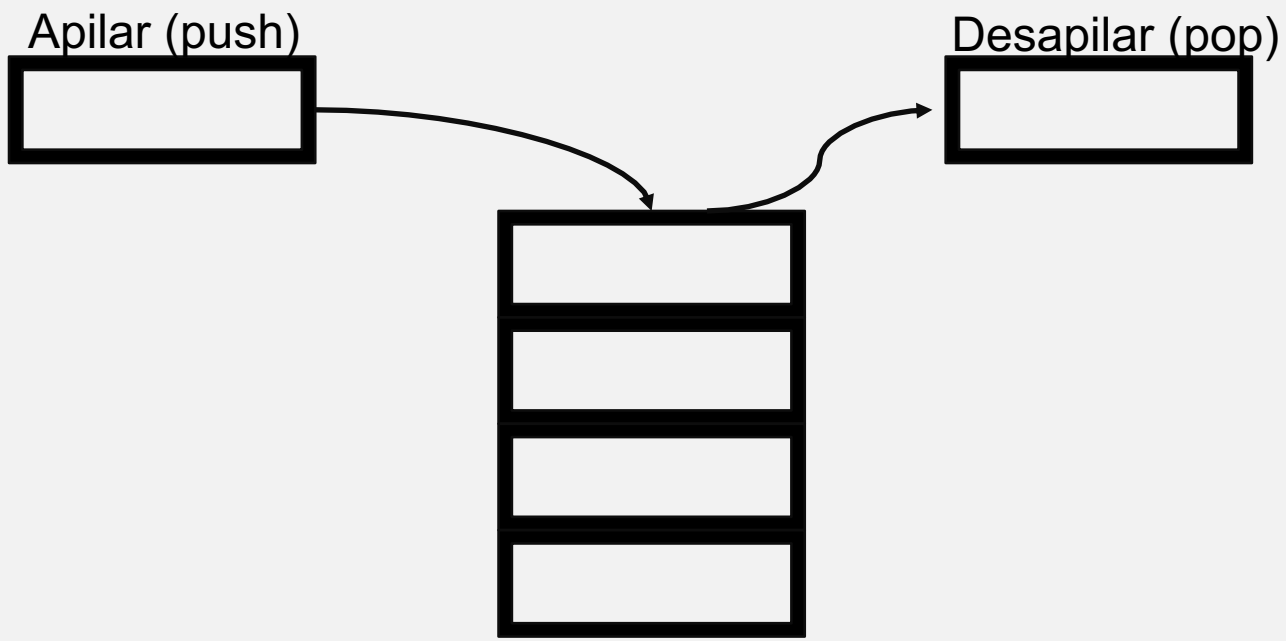
ESTRUCTURA LINEAL : PILA



DEFINICIÓN DE PILA

- Una pila (stack) es una lista ordinal o estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO(Last In First Out) que permite almacenar y recuperar datos
- Las Operaciones fundamentales son:
  - Apilar (push)
  - Desapilar o Retirar (Pop)
- A nivel Lógico una pila es un grupo ordenado de elementos
- El quitar elementos existentes y añadir nuevos puede realizarse solo por la *cabeza*.
- La pila es muy útil en situaciones cuando los datos deben almacenarse y luego recuperarse en orden inverso

REPRESENTACIÓN GRÁFICA DE UNA PILA



1) TAD Pila [T]
2) { invariante : TRUE}
3) Constructoras:
4) crearPila;
5) Modificadoras:
6) apilar: Pila T
7) desapilar: Pila
8) Analizadoras
9) cima: Pila
10) esVacia: Pila
11)Destructora:
12) destruirPila: Pila
13)Pila crearPila(void)
14)/* Crear una pila vacía */
15){ post: crearpila = 𝕆}

16)void apilar(Pila pil,T elem)
17)/* Coloca sobre el tope de la Pila el elemento elem*/
18){ post: pil =e_1, $e_2 \ldots e_{n-1}$}

19)void desapilar(Pila pil)
20)*/Elimina el elemento que encintra en el tope*/}
21){ pre: pil =e_1, $e_2 \ldots e_{n-1}$ N>0}
22){ post: pil =e_1, $e_2 \ldots e_{n-1}$}

23)T cima(Pila pil)
24)*/Retorna el elemento que se encuentra en el tope de la pila*/
25){ pre: n > 0 }
26){ post: cima = en}

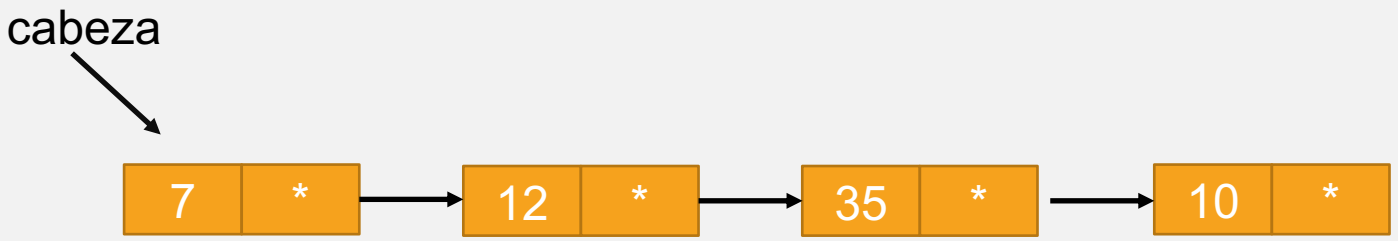
27)int esVacia(Pila pil)
28)/* Informa si la pila esta vacía*/
29){ post: esVacia = (pil == 𝕆) }

30)void destruirPila(Pila pil)
31)/* Detruye la pila retornando toda la memoria ocupada */
32){ post: pil ha sido destruída}

EL TAD PILA

IMPLEMENTACIÓN DE UNA PILA EN JAVA

- Una Pila dinámica que utiliza Nodos enlazados.
- Cada Nodo almacena un valor y contiene una referencia al siguiente Nodo
- Así para construir la siguiente pila, se han aplicado las siguientes operaciones:
  - apilar(10), apilar(35), apilar(12), apilar(7),



Representación de la Pila con Nodos Enlazados

EL TAD PILA

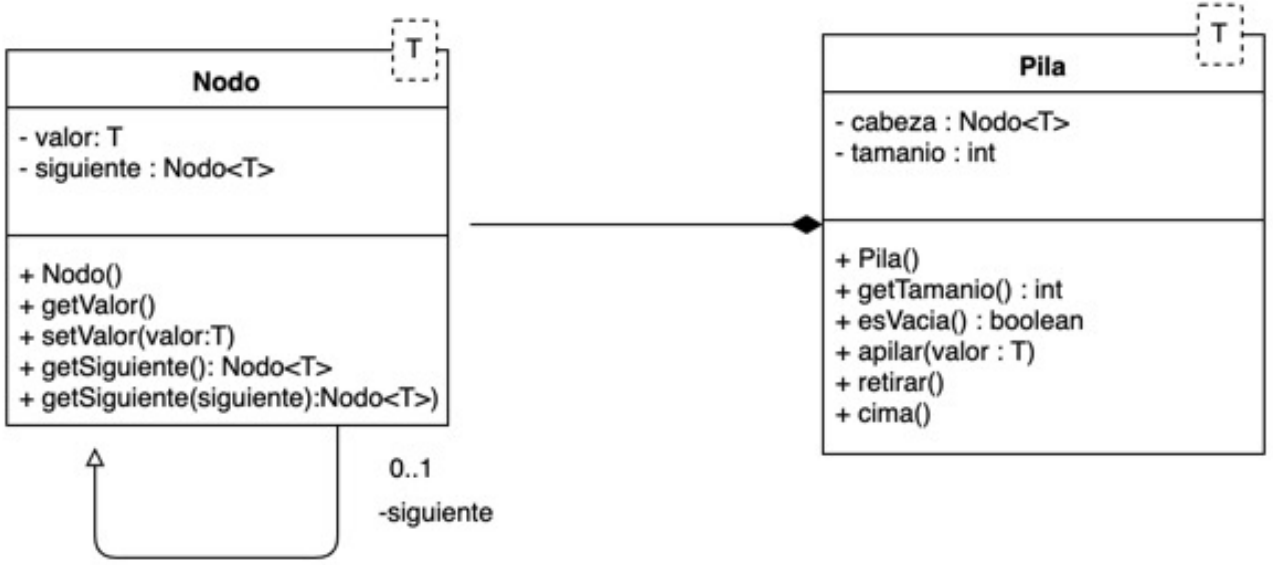


Diagrama de clase de implementación del TAD Pila

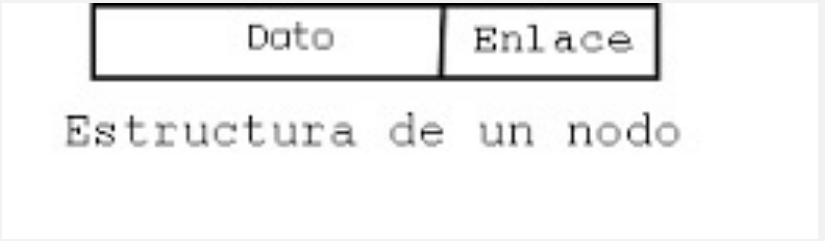
dos clases: *Nodo* y *Pila*
Nodo, representa los nodos enlazados que almacenan los objetos que se apilan.
Dos atributos: *valor* que representa el valor que guardan, en este caso referencia a un objeto tipo T (genérico) y el atributo *siguiente*, representa la referencia al siguiente nodo.
Los demás métodos son contructor, getter y setter de cada atributo

ESTRUCTURA LINEAL: NODO DE UNA PILA

Un NODO se implementa como una clase con dos ATRIBUTOS:

- Uno para contener un dato (Info o dato)
- Y otro para mantener una referencia (ref o Enlace) (o dirección de memoria) hacia el siguiente nodo de la lista.

- La lista enlazada se forma concatenando nodos
- Cada uno de ellos contiene una pieza de información y la dirección de memoria que nos permitirá acceder al siguiente elemento de la colección.
- El primer elemento de la lista enlazada estará referenciado por una variable de tipo puntero.



Estructura de un nodo

ESTRUCTURA NODO

- UN NODO ES UNA ESTRUCTURA QUE SE COMPONE DE UNA PIEZA DE INFORMACIÓN MAS LA DIRECCIÓN DE MEMORIA DE OTRO NODO

CÓDIGO FUENTE CLASE NODO

```
public class Nodo <T>{
   //atributo valor de tipo T.Almacena la referencia al objeto
   //que se guarda en el nodo
   private T valor;
   Nodo<T> siguiente;
   //constructor por defecto
   public Nodo() {
      valor = null;
      siguiente = null;
      }
   public T getValor() {
      return valor;
   }
   public void setValor(T valor) {
      this.valor = valor;
   }
   public Nodo<T> getSiguiente() {
      return siguiente;
   }
   public void setSiguiente(Nodo<T> siguiente) {
      this.siguiente = siguiente;
   }
}
```

CODIGO FUENTE CLASE PILA

```java
package pila;


public class Pila<T> {
   //Atributos cabeza, que apunta al tope de la pila
   private Nodo<T> cabeza;
   //Almacena el total de elementos de la pila
   private int tamanio;
   //Constructor por defecto
   public Pila() {
      cabeza = null;
      tamanio = 0;
    }
   public int getTamanio() {
      return tamanio;
   }
   //Verifica si la pila esta vacia
   public boolean esVacia() {
      return (cabeza == null);
   }
```

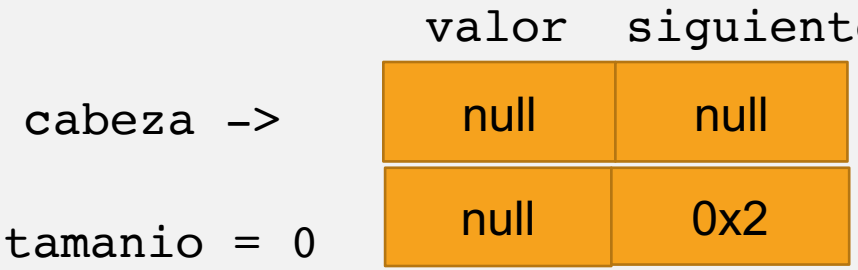
```java
//Apila un elemento nuevo
public void apilar(T valor) {
   //Crear un nuevo nodo
   Nodo<T> nuevo = new Nodo<T>();
   //Fijar el valor dentri de nuevo
   nuevo.setValor(valor);
   if (esVacia()) {
      //cabeza apunta al nuevo nodo
      cabeza = nuevo;
   }
   else {
      //se enlaza el campo siguiete de nuevo con    la cabeza
      nuevo.setSiguiente(cabeza);
      //la nueva cabeza de la pila pasa a ser nuevo
      cabeza = nuevo;
   }
   //Incrementamos el tamaño de la pila
   tamanio++;
}
```

CODIGO FUENTE CLASE PILA

```
    //Elimina un elemento de la pila
    public void retirar() {
        if (!esVacia()) {
            cabeza = cabeza.getSiguiente();
            tamanio--;
        }
    }
    //Devuelve el elemento almacenado en el     tope de la pila
    public T cima() {
        if(!esVacia())
            return cabeza.getValor();
        else
            return null;
    }
}
```

PASO A PASO PROCESO APILAR

1.- Crear un nuevo objeto tipo Pila



tamanio = 0

2.- Invocar al método apilar del objeto tipo Pila con apilar(5)

tamanio = 1

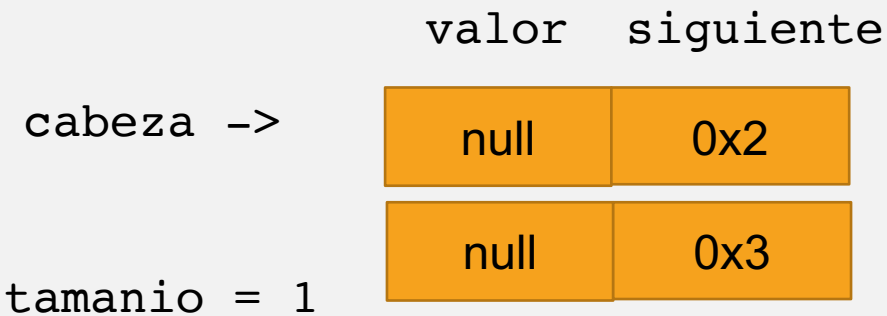
3.- Invocar al método apilar del objeto tipo Pila con apilar(8)

Si la pila esta vacía entonces -> cabeza = nuevo

Si la pila NO vacía entonces -> nuevo.setSiguiente(cabeza) y cabeza=nuevo;

PASO A PASO PROCESO APILAR

1.- Crear un nuevo objeto tipo Pila

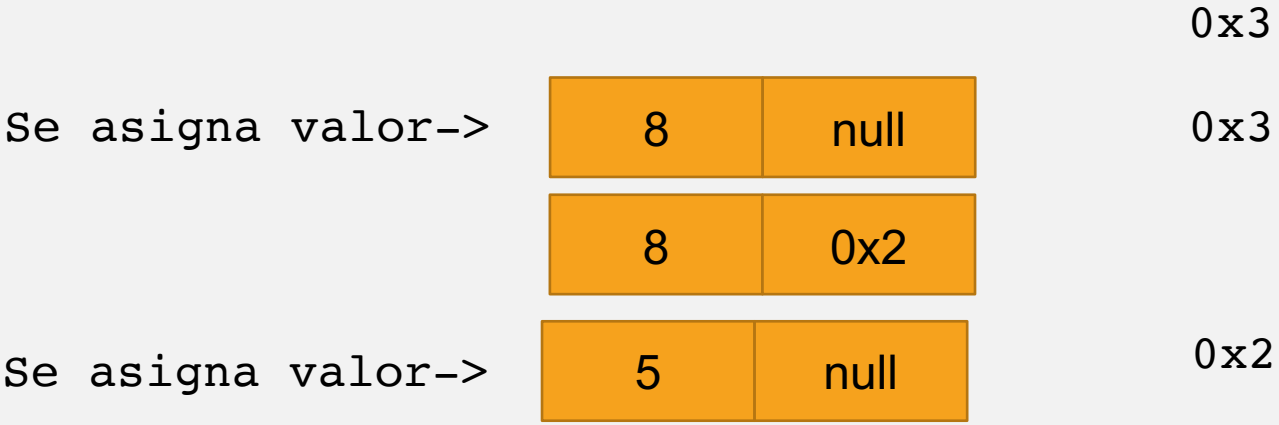


2.- Invocar al método apilar del objeto tipo Pila con apilar(5)

tamanio = 1

tamanio = 2

3.- Invocar al método apilar del objeto tipo Pila con apilar(8)

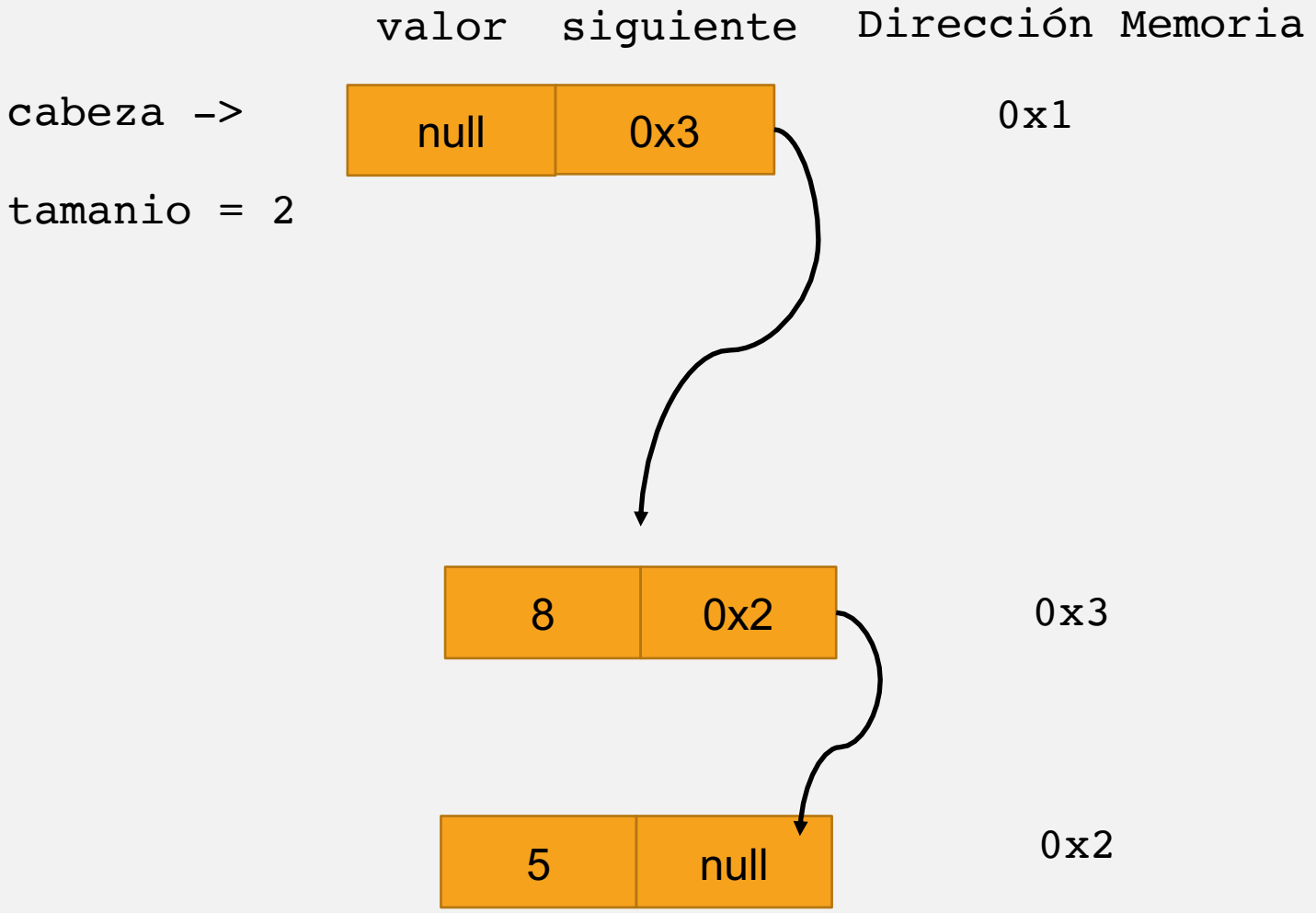


Si la pila esta vacía entonces -> cabeza = nuevo

Si la pila NO vacía entonces -> nuevo.setSiguiente(cabeza) y cabeza=nuevo;

PASO A PASO PROCESO APILAR

1.- Un objeto Pila con dos Nodos



LA CLASE STACK DE JAVA

- LA API de Java ya trae implementa una Clase Stack y esta en *java.util.Stack.*
- Las operaciones básicas son: *push, pop, peek, empty, search*
- *push.* Introduce un nuevo elemento en la pila
- *pop. Saca un elemento de la pila*
- *peek. C*onsulta el primer elemento de la cima de la pila
- *empty. C*omprueba si la pila esta vacía
- *search. B*usca un determinado elemento dentro de la pila y devuelve su posición

EJEMPLO DEL USO DE LA CLASE STACK

```
package pila;


import java.util.Stack;


public class EjemploStack {


    public static void main(String[] args) {
    //Crea una nueva pila de enteros
    Stack<Integer> pila = new Stack<Integer>();
    //Se apila algunos datos enteros
    pila.push(2);
    pila.push(5);
    pila.push(7);
    System.out.println("El tope de la pila es: "+
            pila.peek());
    //Se desapila un elemento
    pila.pop();
    System.out.println("El tope de la pila es: "+
            pila.peek());
    //Se desapila un elemento
    pila.pop();
    System.out.println("El tope de la pila es: "+
    pila.peek());
    //Se desapila un elemento
    pila.pop();
    //Como la pila esta vaia dispara una    excepción
    System.out.println("El topo de la pile es: "+
    pila.peek());
    }
}
```

PROBLEMAS QUE SE RESUELVEN CON PILA

- Evaluación de la correspondencia de delimitadores en un programa

```
while (m<(n[8]+o)) {
int p=7;
/*comentarios*/
}
```

- Algoritmo que lo resuelva:

1. Obtener el carácter de la expresión y repetir los pasos 2 al 3
2. Si es un operador de apertura: (,[,/* se lo apila
3. Si es un operador de cierre:),],*/:
    1. Comparar que corresponda con el operador del tope de la pila
    2. Si no corresponde, termina el programa y la expresión es incorrecta
    3. Si hay correspondencia, se elimina el elemento del tope de la pila y vover a paso 1
4. Si al terminar de evaluar la expresión quedan elementos en la pila, la expresión es incorrecta
5. Si al terminar de evaluacir la expresión de la pila queda vacía, la expresión es correcta
6. Fin del algoritmo.

```java
package pilas;



public class Delimitadores {

//Evalua si una cadena tiene los delimitadores correctos

public  boolean evaluacionDelimitadores(String cadena) {
   Pila<String> pcaracteres = new Pila<String>();


   int i = 0;
   boolean masElementosPorLeer = true;
   while (i<cadena.length() && masElementosPorLeer)
   {
      char car = cadena.charAt(i);
      String s = charToString(car);
      i++;
      switch (car)
{
//En caso de Apilar
case '(':{
    pcaracteres.apilar(charToString(')'));
    break;
}
case '[':{
   pcaracteres.apilar(charToString(']'));
   break;
}

case '{':{
   pcaracteres.apilar(charToString('}'));
   break;
}
case '/':{
   if (siguienteEsAsterisco(cadena,i)) {
      pcaracteres.apilar(charToString('/'));
   i++;
   }
   break;
}
```

SOLUCIÓN DELIMITADORES

https://replit.com/@LourdesArmenta/Pilas?v=1

```
//Para Desapilar
case ')':
case ']':
case '}':
{
   //comparar que corresponda
   String aux = pcaracteres.cima();
   if (aux!=null)
   {

   if(s.compareTo(aux)==0)
   {
      //hay correspondencia por lo tanto
      //lo elimino
      pcaracteres.retirar();
   }
   else
   {
      masElementosPorLeer = false;
   }
 }
```

```
else
   {
      masElementosPorLeer = false;
   }
}
case '*':{
   if (siguienteEsDiagonal(cadena,i)) {
      pcaracteres.retirar();
      i++;
   }
 }
}
}
//Si la Pila aún tiene elementos es un Error
if (pcaracteres.esVacia() &&   masElementosPorLeer)
   return true;
else
   return false;
} //Termina método
```

```
private static boolean
siguienteEsAsterisco(String cadena, int
posicion) {
   char car = cadena.charAt(posicion);
   if (car=='*')
      return true;
   else
   return false;
}
private static boolean
siguienteEsDiagonal(String cadena, int
posicion) {
   char car = cadena.charAt(posicion);
   if (car=='/')
      return true;
   else
      return false;
}
```

```
//convierte un char a un objeto tipo
String
private static String charToString(char
ch) {
  return String.valueOf(ch);
}



}//Termina clase
```

EVALUACIÓN DE EXPRESIONES ARITMÉTICAS

- Las pilas se utilizan para evaluar expresiones aritméticas. Por ejemplo:

$$\frac{(100 + 23) * 231}{(31 - 14)^2} - (34 - 12)$$

Se debe pasar a expresiones en *notación postfija* y aplicar un algoritmo para la evaluar la expresión en notación postfija

La expresion

A + B es *notació infija*

AB+ es notación postfija

+AB es notación prefija

- *Ventaja de Postfija, no hay que utuilizar paréntesis, para indicar el orden de operación, ya que queda establecido por la ubicación de los operadores con respecto a los operandos, ejemplo:*
- *(X + Z) * W / T ^ Y - V (INFIJA)*
- *X Z + W * T Y ^ / V - (POSTFIJA)*

PARA CONVERTIR DE INFIJA A POSTFIJA HAY CIERTAS CONDICIONES

- SOLAMENTE SE MANEJARAN LOS SIGUIENTES OPERADORES (DE MAYOR A MENOS) SEGÚN SU PRIORIDAD DE EJECUCIÓN:

PRIORIDAD DE OPERADORES		
OPERADOR	PRIORIDAD DE LA PILA	PRIORIDAD FUERA DE LA PILA
^: Potencia	3	3
*/:Multiplicación y División	2	2
+-:Suma y Resta	1	1
(:Paréntesis izquierdo	0	4

- Los operadores de más alta prioridad se ejecutan primero. Si hubiera en una expresión dos o más operadores de igual priodidad, entonces se procesan de izquierda a derecha. Las subexpresiones parentizadas tendrán más prioridad que cualquier operados.
- Obsérvese que no se trata el paréntesis derecho ya que éste provoca sacar operadores de la pila hasta el paréntesis izquierdo.

ALGORITMO PARA CONVERTIR UNA EXPRESIÓN INFIJA A POSTFIJA

- Se parte de una expresión en notación infija
- Se utiliza una pila para almacenar operdores y los paréntesis izquierdo
- La expresión se lee de izquierda a derecha, carácter por carácter, los operandos pasan directamente a formar parte de la expresión postfija, que se guarda en un arreglo.
- Los operadores se meten en la pila siempre que éste vacía, o bien siempre que tenga mayor prioridad que el operados de la cima de la pila (o bien si es la máxima prioridad)
- Si la prioridad es menor o igual se saca el elemento cima de la pila y se vuelve a hacer comparación con el nuevo elemento de la cima
- Los paréntesis izquierdo siempre se menten en la pila; dentro de la pila se les considera de mínima prioridad para que todo operador que se encuentra dentro del paréntesis entre en la pila.
- Cuando se lee un paréntesis derecho hay que sacar todos los operadores de la pila pasando a formar parte de la expresión postfija, hasta llegar a un paréntesis izquierdo, el cual se elimina ya que los paréntesis no forman parte de la expresión postfija
- El proceso termina cuando no hay mas elementos de la expresión y la pila esta vacía

ALGORITMO PARA CONVERTIR UNA EXPRESIÓN INFIJA A POSTFIJA

1. Obtener caracteres de la expresión y repetir paso 2 al 5 para cada carácter
2. Si es un operando pasarlo a la expresión postfija
3. Si es un operador:
    1. Si la pila está vacía, meterlo a la pila. Repetir a partir del paso 1
    2. Si la pila no esta vacía:
        1. Si la prioridad del operador leìdo es mayor que la prioridad del operador cima de la pila, meterlo en la pila y repetir a partir del paso 1
        2. Si la prioridad del operador es menor o igual que la prioridad del operador de la cima, sacar operador cima de la pila y pasarlo a la expresión postfija. Volver a paso 3
4. Si es paréntesis derecho:
    1. Sacar operador cima de la pila y pasarlo a la expresión postfija
    2. Si nueva cima es paréntesis izquierdo, suprimir elemento cima
    3. Si cima no es paréntesis izquierdo volver a paso 4.1
    4. Volver a partir del paso 1
5. Si es paréntesis izquierdo pasarlo a la pila
6. Si quedan elementos en la pila, pasarlos a la expresión postfija
7. Fin del algoritmo.

EVALUACIÓN DE LA EXPRESIÓN EN NOTACIÓN POSTFIJA

1. Una vez convertida la expresión infija en postfija, aplicar un Algoritmo (también menejando una pila) para evaluar la expresión
2. Para ello se almacena la expresión artimética transformada a notación postfija en un vector, en la que los operandos estan representados por variables de una sola letra. Antes de evaluar la expresión requiere dar valores numéricos a los operandos. Una vez que se tienen los valores de los operandos, la expresión es evaluada.

ALGORITMO DE EVALUACIÓN DE LA EXPRESIÓN EN NOTACIÓN POSTFIJA

1. Examinar el vector desde el elemento 1 hasta el N, repetir los pasos 2 y 3 para cada elemento del vector.
2. Si el elemento es un operando meterlo en la pila
3. Si el elemento es un operador, lo designamos por ejemplo con +:
    1. Sacar los dos elementos superiores de la pila, los denominamos con los identificadores x,y repectivamente
    2. Evaluar x+y; el resultado es z = x + y
    3. El resultado z, meterlo en la pila
    4. Repetir a partir del paso 1.
4. El resultado de la evaluación de la expresión está en el elemento de la pila
5. Fin del algoritmo

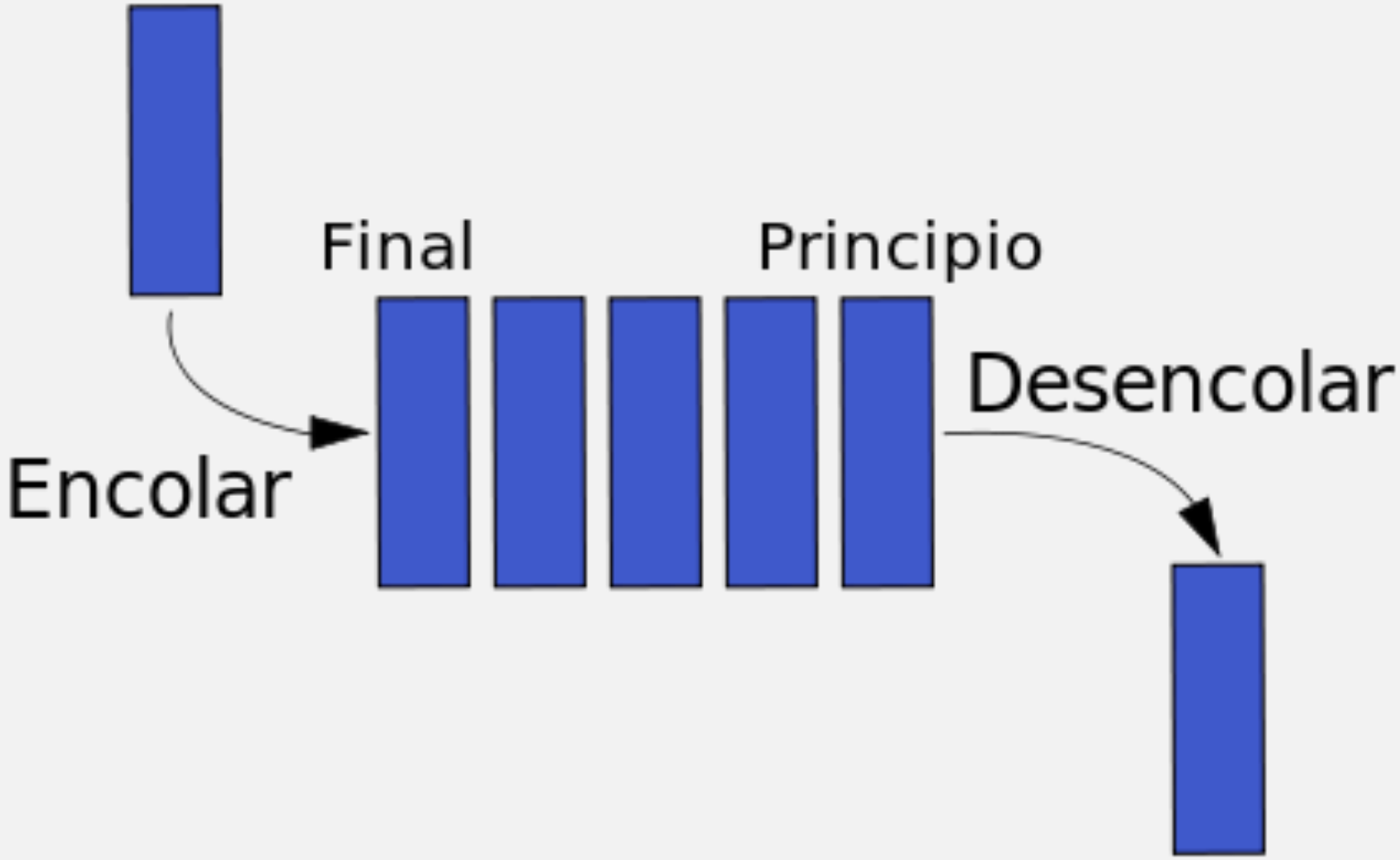
EJEMPLO

- Sea la expresión postfija XZ+W*TY^/V-
- Haciendo X=3,Z=1,W=6,T=2,Y=3,V=1
- Obtenemos: 3 1 + 6 * 2 3 ^ / 1 –

Comprobar $\frac{(3+1)*6}{2^3} - 1 = 2$

Evaluando la expresión Postfija		
Paso	Elemento Leído	Pila
1	3	3
2	1	3 1
3	+	4
4	6	4 6
5	*	24
6	2	24 2
7	3	24 2 3
8	^	24 8
9	/	3
10	1	3 1
11	-	2

ESTRUCTURA LINEAL: COLA



DEFINCION DE COLA

- ES UNA ESTRUCTURA INVERSA A LA PILA
- LA FILOSOFÍA ES FIFO (FIRST IN, FIRST OUT)
- SE CARACTERIZA POR SE UN SECUENCIA DE ELEMENTOS EN LA QUE LA OPERACIÓN DE INSERCIÓN SE REALIZA POR UN EXTREMO (ENCOLAR) Y LA OPERACIÓN DE EXTRACCIÓN POR EL OTRO (DESENCOLAR)

IMPLEMENTACIÓN DE UNA COLA EN JAVA

1. Ver práctica en clase
2. Videos consultados por los alumnos:
   - https://www.youtube.com/watch?v=5CClpYQTGUI
   - https://www.youtube.com/watch?v=6i2f6k5PPjs
   - https://www.youtube.com/watch?v=TiCG0JC-PYM

EL TAD COLA

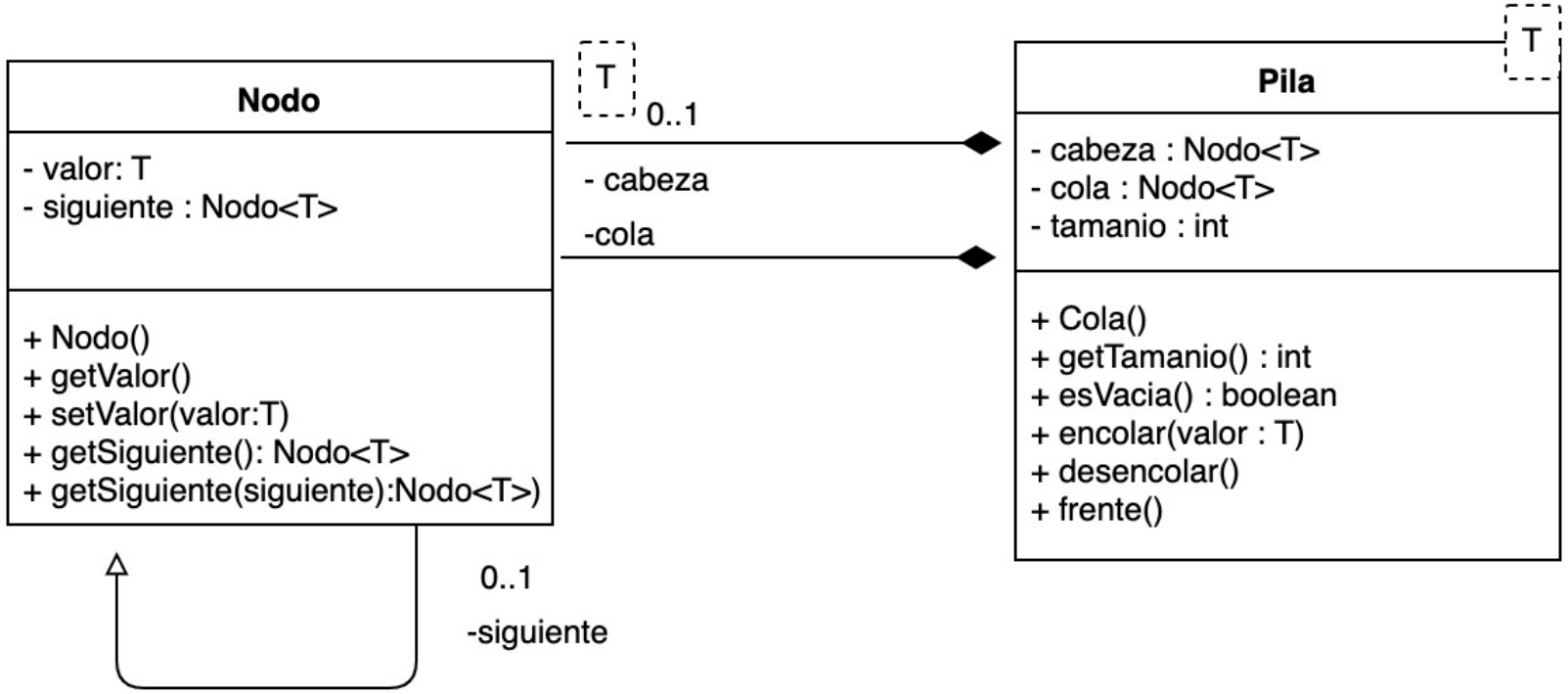


Diagrama de clase de implementación del TAD Pila

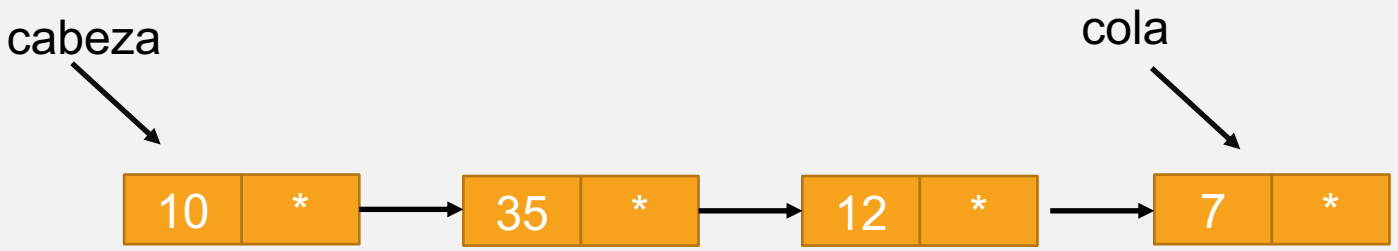
```
1) TAD Cola [T]
2) { invariante : TRUE}
3) Constructoras:
4)     crearCola;
5) Modificadoras:
6)    encolar: Cola T
7)    desencolar: Cola
8) Analizadoras
9)    frente: Cola
10)   esVacia: Cola
11)Destructora:
12)   destruirCola: Cola
13)Cola crearCola(void)
14)/* Crear una Colavacía */
15){ post: crearcola = 𝕆}

16)void encolar(Cola col,T elem)
17)/* Agregar elem al final de la
   cola*/
18){ post: col =e1, e2…en-1}
```

```
19)void desancolar(Cola col)
20)*/Elimina el primer elemento de la
   cola */}
21){ pre: N>0}
22){ post: col =e1, e2…en-1}

23)T frente(Cola col)
24)*/Retorna el elemento de la cola*/
25){ pre: n > 0 }
26){ post: frente = e1}

27)int esVacia( Cola col)
28)/* Informa si la cola esta vacía*/
29){ post: esVacia = (col == 𝕆) }

30)void destruirCola(Cola col)
31)/* Detruye la cola retornando toda la
   memoria ocupada */
32){ post: col ha sido destruída}
```

EL TAD COLA

IMPLEMENTACIÓN DE UNA COLA EN JAVA

- Una Cola dinámica que utiliza Nodos enlazados.
- Cada Nodo almacena un valor y contiene una referencia al siguiente Nodo
- Así para construir la siguiente cola, se han aplicado las siguientes operaciones:
  - encolar(10), encolar(35), encolarr(12), encokar(7).
- Cada nodo almacena un dato y a la vez una referencia que almacena la dirección del siguiente nodo.
- Existe una referencia llamada cabeza que apunta al primer elemento que entro (10) y otra referencia que apunta al último dato que entro (7)



Representación de la Cola con Nodos Enlazados

CODIGO FUENTE CLASE COLA

- https://replit.com/@LourdesArmenta/Cola

LA INTERFASE QUEUE DE JAVA

- La API de Java tiene la interface java.util.Queue
- Una interface se requiere instanciar como un objeto concreto
- Por lo que se puede implementar con la API Collections de java
  - java.util.LinkedList
  - java.util.PriorityQueue
- LinkedList es una cola estándar
- PriorityQueue es una cola de prioridades que almacena sus elementos internos de acuerdo a un orden
- Las operaciones básicas de Queue son:
  - add() – Inserta un elemento en la cola
  - remove() – devuelve y remueve el primer elemento de la cola
  - peek() – devuelve el primer elmento de la cola y devuelve null cuando la cola esta vacía

EJEMPLO DEL USO DE LA INTERFACE QUEUE DE JAVA

```
package cola;
/* Ejemplo del uso de la interfase Queue*/
import java.util.Queue;
import java.util.LinkedList;


public class ClienteQueue {
   public static void main(String[] args) {
   //Crea una cola genérica
   Queue micola = new LinkedList();
   //Agrega tres elemenos a la cola en ese
orden
   micola.add("elemento 0");
   micola.add("elemento 1");
   micola.add("elemento 2");
   //muestra el primer elemento de la cola
   System.out.println("El primer elemento
dela cola es"
   + micola.peek());
      micola.remove();
      System.out.println("El primer elemento
dela cola es"
      + micola.peek());
      //elimina el último elemento de la cola
      micola.remove();
      //Imprime null porque esta vacia
      System.out.println("El primer elemento
dela cola es"
      + micola.peek());
   }//fin de main
} // fin de clase
```

PROBLEMAS QUE SE RESUELVEN CON COLAS

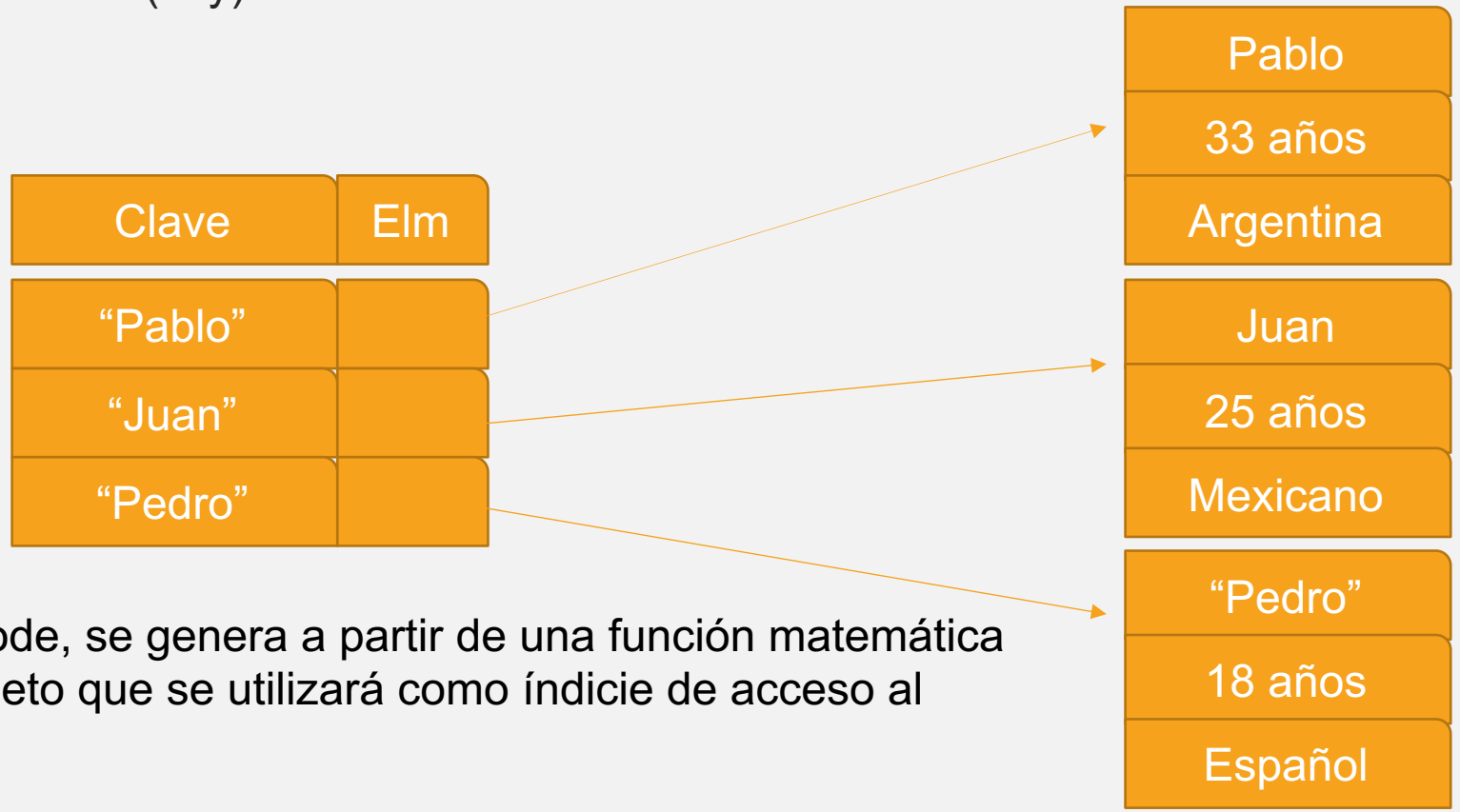
- Las colas se utilizan en sistemas informáticos, transportes y operaciones de investigación, etcétera, donde los objetos, personas o eventos son tomados como datos que se almacenan y se guarden mediante colas para su posterior procesamiento.
- Ejemplo: Simulador del despliegue y aterrizague de aviones
  - Se debe utilizar una cola para simular el despegue y aterrizaje de aviones, las polñiticas son:

POLITICAS DE LA SIMULACION DE DESPEGUE Y ATERRIZAJE

1. Se tiene estimado que del aeropuerto puden salir vuelos cada 5 minutos, siempre y cuando no haya solicitudes de aterrizaje
2. Un vuelo no puede salir del aeropuerto, antes de la hora programada
3. Un vuelo no puede despegar si se encuentran vuelos pendientes por despejar, es decir deben respetar la cola
4. El aeropuerto conoce con anterioridad la hora de salida de todos los vuelos programados para ese día. Las aerolineas tienen programadas vuelos con 10 minutos de diferencia
5. Los aterrizajes ocurren de manera aleatoria. El aterrizaje dura diez minutos (es prioridad el aterrizaje)
6. Se asume que la simulación inicia con mínimo 10 solicitudes de despegue.

TABLAS DE DISPERSION (HASHTABLE)

- Es una estructura de datos que permite mantener elementos asociados a una determinada clave (usualmente un string). Luego el acceso a cada elemento se hará especificando el valor de esta clave (key) de acceso



hashCode, se genera a partir de una función matemática que objeto que se utilizará como índicie de acceso al array

ESTRUCTURA DE DATOS COMBINADAS

Esta estructura representa a una hashtable donde cada elementos es una lista enlazada de (en este caso) personas
Con el mismo nombre (clave)

LISTA

Una lista enlazada consiste en una secuencia de nodos, en los que se guardan campos de datos arbitrarios y uno o dos referencias al nodo anterior y/o superior.

Las listas enlazadas tienen la particularidad de que las **inserciones** y **extracciones** se realizan en **cualquier parte** de la lista.

LA DIFERENCIA ENTRE LAS TRES ESTRUCTURAS VENDRÁ DADA POR LA POSICIÓN DEL ELEMENTO A AÑADIR, BORRAR Y CONSULTAR:

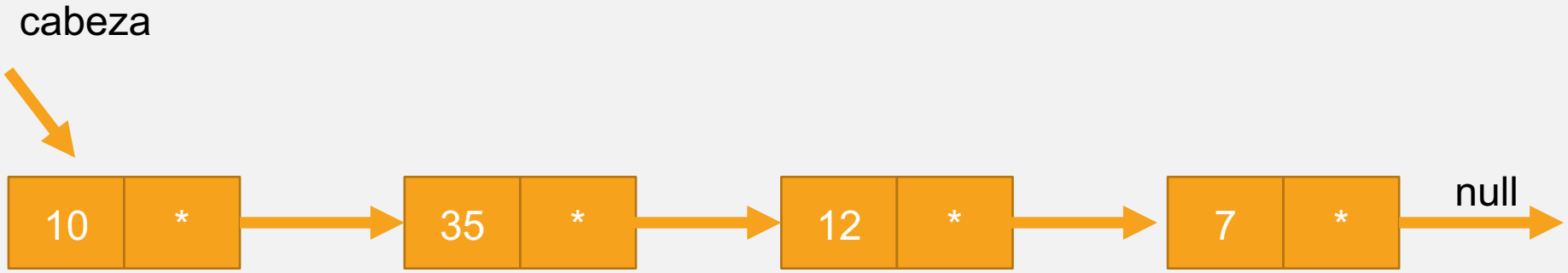
- Pilas: las tres operaciones actúan sobre el final de la secuencia
- Colas: se añade por el final y se borra y consulta por el principio
- Listas: las tres operaciones se realizan sobre una posición privilegiada de la secuencia, la cual puede desplazarse

TIPOS DE LISTAS

- Listas enlazadas simples
- Listas doblemente enlazadas
- Listas circulares
- Listas doblemente enlazadas circulares
- Listas de listas

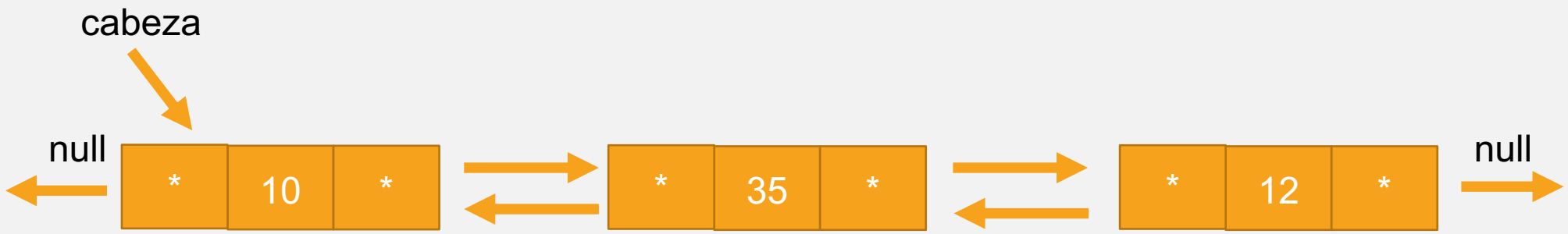
LISTAS ENLAZADA SIMPLE

- Consisten en una secuencia de nodos, donde existe un *enlace por nodo*. Este Enlace apunta al siguiente nodo de la lista, o al valor Nulo, si es el úmtimo nodo



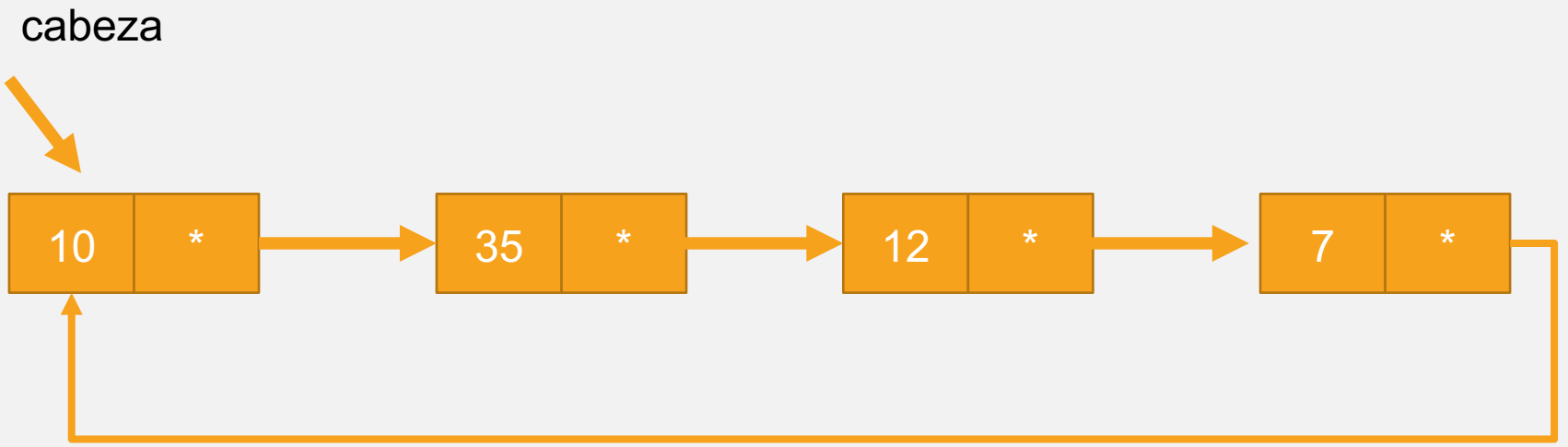
LISTAS DOBLEMENTE ENLAZADA

- *Cada nodo* tiene *dos enlaces*: uno apunta al nodo anterior, o apunta al valor Nulo si es el primer nodo; y otro que apunta al seguiente nodo, o apunta al valor Nulo si es el último Nodo

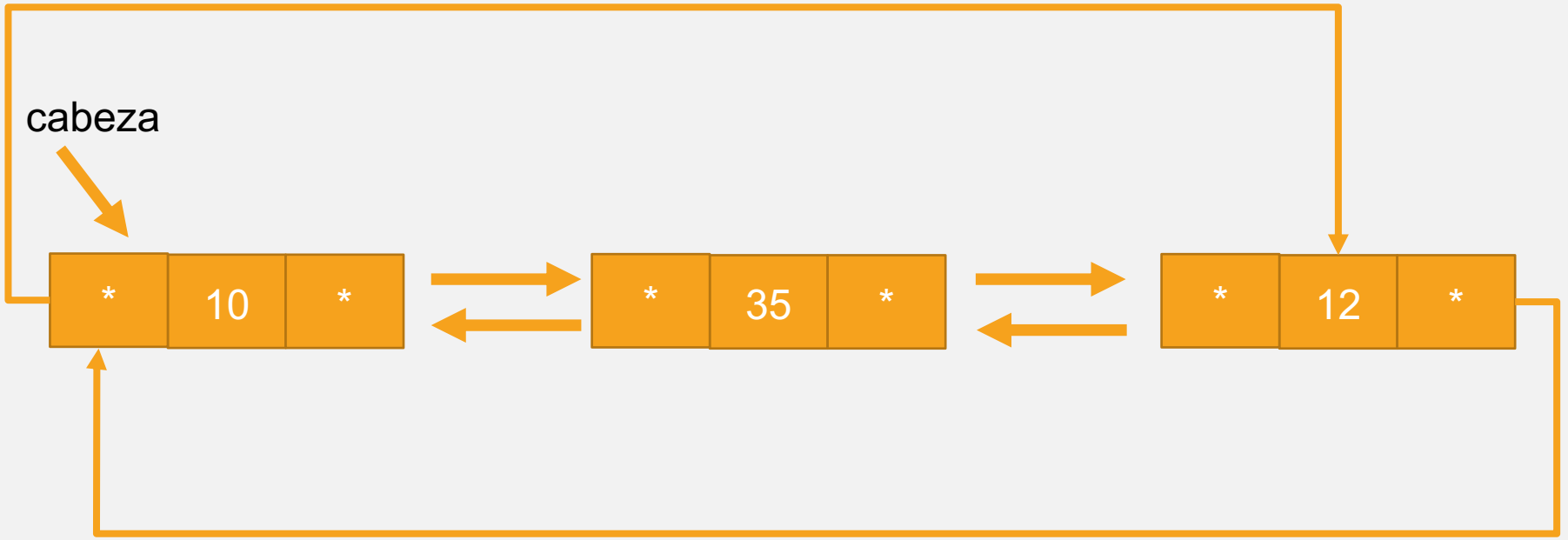


LISTAS CIRCULARES

- El primer y el último nodo están unidos.



LISTAS DOBLEMENTE CIRCULARES



IMPLEMENTACIÓN DEL TAD LISTA EN JAVA

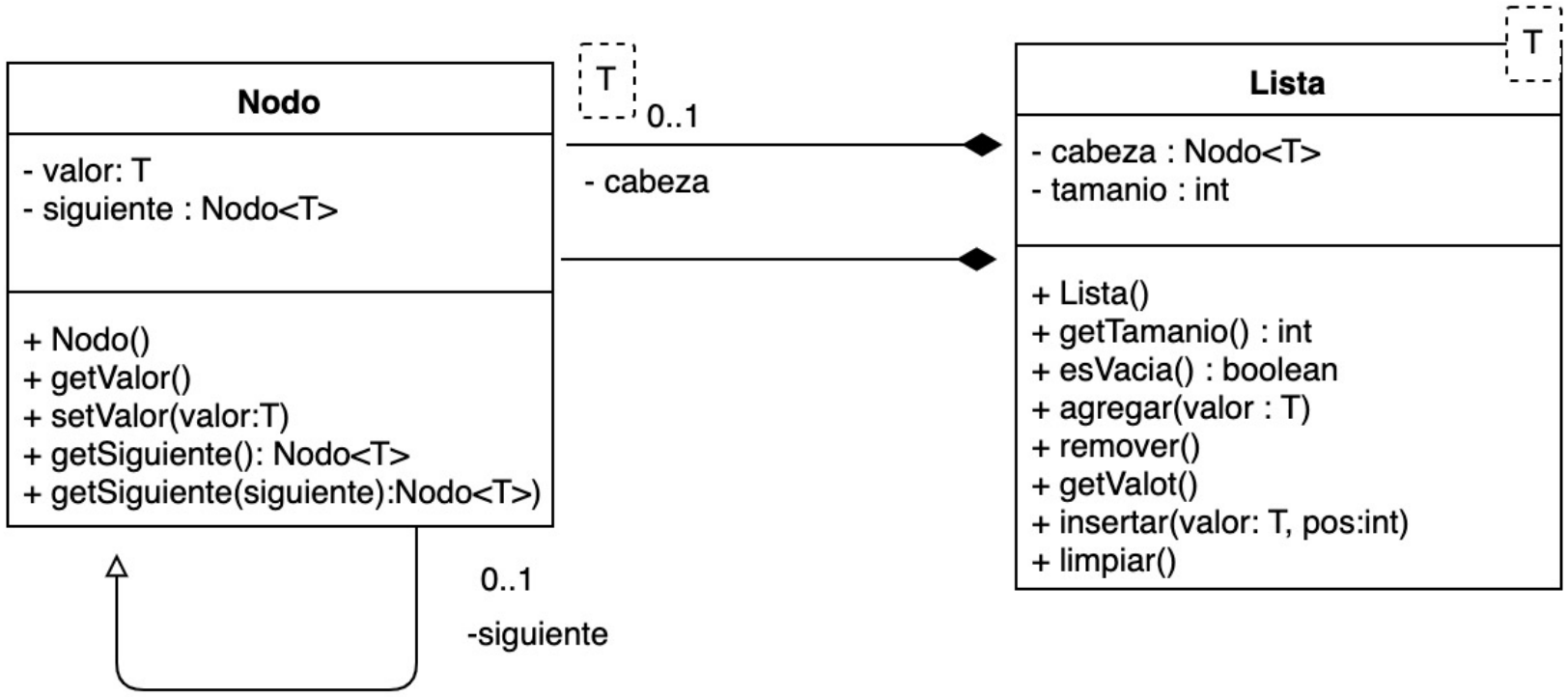


Diagrama de clase de implementación del TAD Lista

1) TAD Lista [T]
2) { invariante : TRUE}
3) Constructoras:
4) crearLista;
5) Modificadoras:
6) agregar: Lista T
7) insertar: Lista T int
8) remover: Lista int
9) Analizadoras
10) getValor: Lista int
11) setVacia: Lista
12)Destructora:
13) destruirLista: Lista
14)Cola crearLista(void)
15)/* Crear una Lista vacía */
16){ post: crearLista = ∅}

17)void agregar(Lista lis,T elem)
18)/* Agregar elem al final de la lista*/
19){ post: list =e_1, e_2…elem}

19)void insertar(Lista list,T elem,int i)
20)*/agrega elem en la posición i de la lista */}
21){ pre: i< i<n}
22){ post: list =e_1,}

23)void remover(Lista list,int i)
24)*/elimina de la lista de la posicion i*/
25){ pre: i< i<n}
26){ post: list = e $e1$}

27)T getValor(Lista list, int i)
28)/* Retorna el valor almacenadp en la posición i*/
29){ pre: i< i<n}
30){ post: getValor = e_i}
31)void esVacia(Lista list)
32)/* Informa si la listsa esta vacía*/
33){ post: esVacia =(list= ∅)}
34)void destruirLista (List list)
35)/* Destruye la lista retornando toda la memoria ocupada *
36){ post: list ha sido destruída}

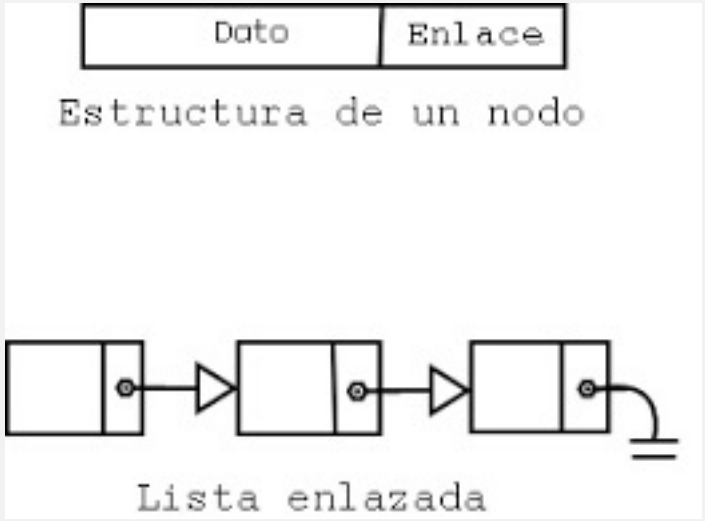
EL TAD LISTA

CODIGO FUENTA LISTA SIMPLEMENTE LIGADA

- https://replit.com/@LourdesArmenta/Lista

ESTRUCTURA LINEAL: NODO DE UNA LISTA ENLAZADA

- La lista enlazada se forma concatenando nodos
- Cada uno de ellos contiene una pieza de información y la dirección de memoria que nos permitirá acceder al siguiente elemento de la colección.
- El primer elemento de la lista enlazada estará referenciado por una variable de tipo puntero.



EJEMPLO DE UNA LISTA SIMPLEMENTE LIGADA (AGREGAR)

- Crear una Lista

cabeza=null

tamanio=0

- agregar 12 (como la lista, esta vacía) solo

cabeza=

tamanio= 1

Crear Nodo=

0

agregar 15 (agregar al final) recorro iniciando por la cabeza

cabeza=

tamanio= 2

aux = cabeza

Crear Nodo=

0

1

agregar 20 (agregar al final) recorro iniciando por la cabeza

cabeza=

tamanio= 3

aux = cabeza

Crear Nodo=

0

1

2

EJEMPLO DE UNA LISTA SIMPLEMENTE LIGADA (INSERTAR)

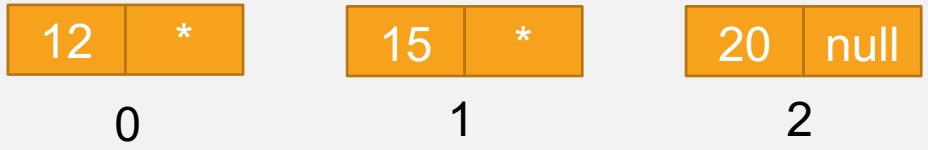
insertar 13 (en la posición 1) en medio de la lista

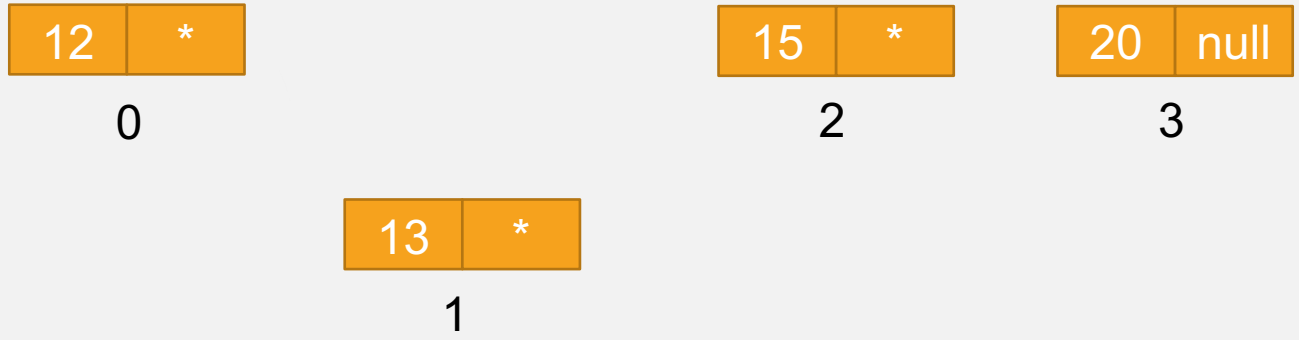
cabeza=

tamanio= 4

aux = cabeza

Crear Nodo= 





EJEMPLO DE UNA LISTA SIMPLEMENTE LIGADA (INSERTAR)

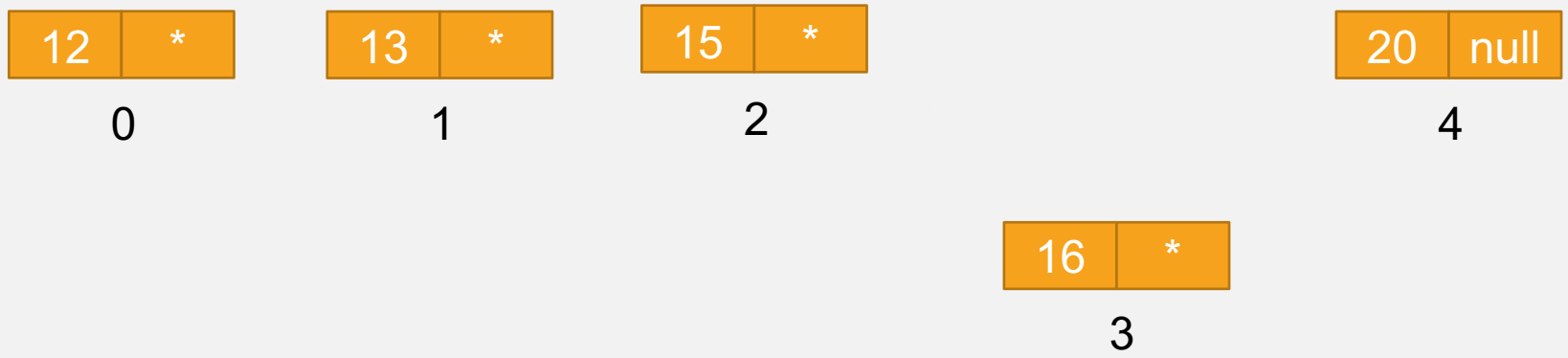
insertar 16 (en la posición 3) en medio de la lista

cabeza=

tamanio= 5

aux = cabeza

Crear Nodo= 



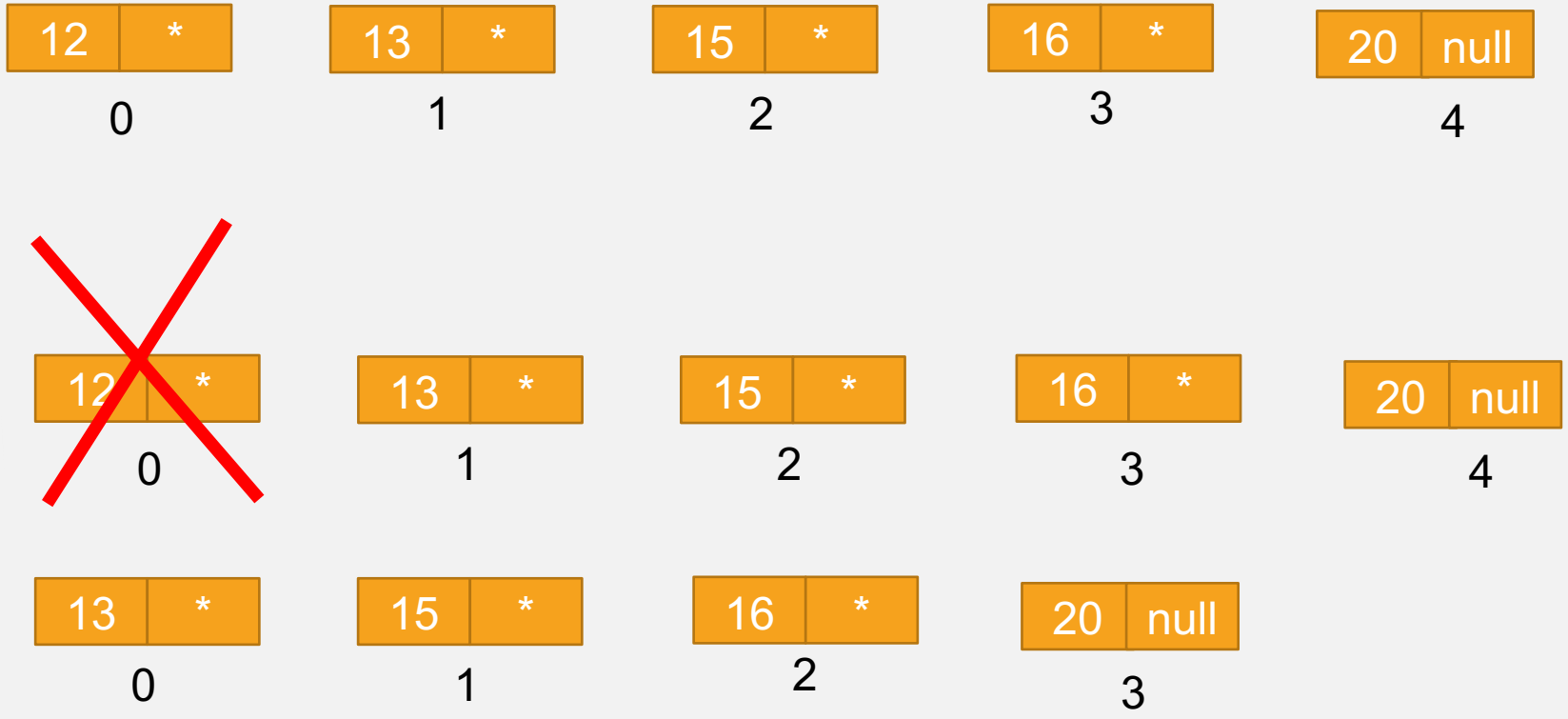
EJEMPLO DE UNA LISTA SIMPLEMENTE LIGADA (REMOVER)

remover (posicion 0) al inicio de la lista

cabeza=

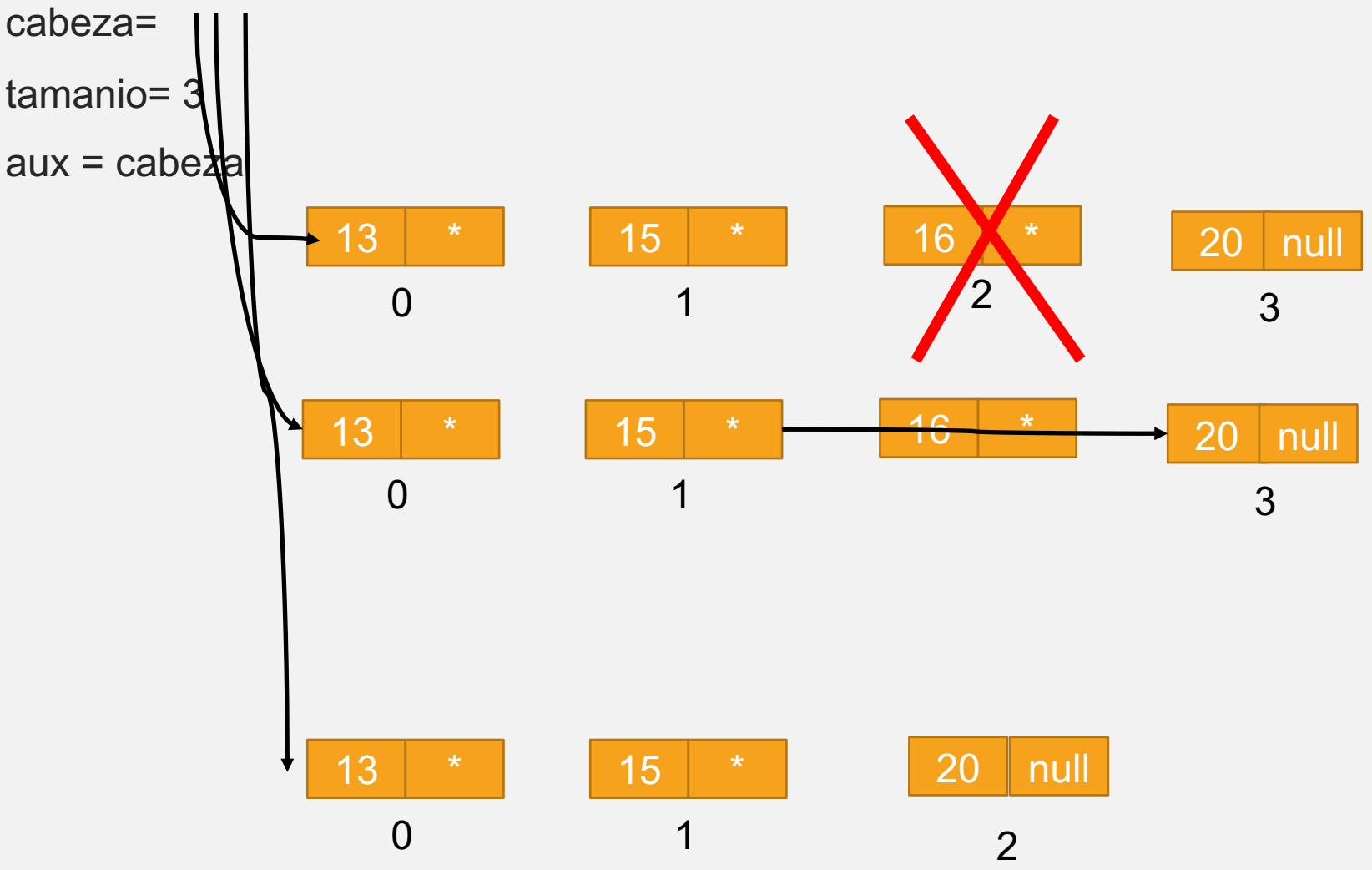
tamanio= 4

aux = cabeza



EJEMPLO DE UNA LISTA SIMPLEMENTE LIGADA (REMOVER)

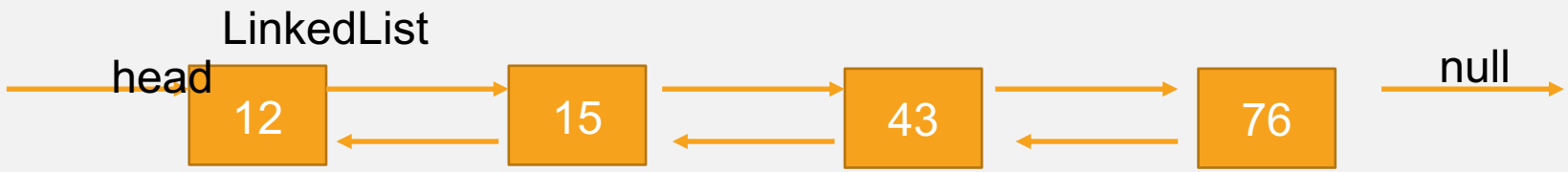
remover (posicion 3) En medio de la lista

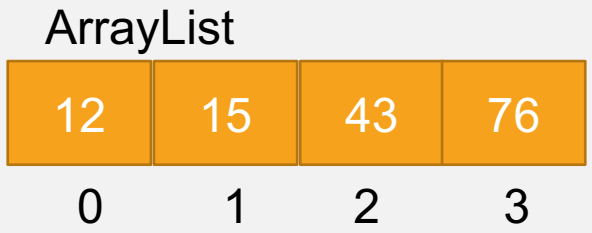


LAS CLASES ARRAYLIST Y LINKEDLIST DE JAVA

- La API de java ya trae implementada las listas mediante las clases ArrayList y LinkedList. Ambas implementan la interfaz genérica: la interfaz List. Eso quiere decir que tendrán una serie de métodos comunes propios de esa interfaz
- LinkedList y ArrayList son dos implementaciones diferentes de la interfaz List. LinkedList usa internamente una lista doblemente enlazada, mientras que ArrayList usa un arreglo que cambia de tamaño dinámicamente

DIFERENCIA ENTRE LINKEDLIST Y ARRAYLIST





LinkedList permite eliminar e insertar elementos en tiempo constante usando iteradores, pero el acceso es secuenc
por lo que encontrar un elementos toma un tiempo proporcional al tamaño de la lista. Normalmente la complejidad
Esa operación promedio sería O(n/2) sin embargo usa una lista doblemente ligada el recorrido puede ocurrir desde
principio o el final de la lista por lo tanto resulta en O(n/4).

ArrayList ofrece acceso en tiempo constante O(1), pero si se quiere añadir o remover un elemento en cualquier pos
que no sea la última es necesario mover elementos. Además si el arreglo ya está lleno es necesario crear uno nue
mayor capacidad y copiar los elementos existentes.

OPERACIONES QUE MANEJAN LINKEDLIST Y ARRAYLIST

- add(Object) : Agregar un elemento al final
- add(int,Object): Agrega un elemento en la posición especificada en el primer argumento.
- clear() : Elimina todos los elementos.
- get(int): Devuelve el elemento de la posición especificada.
- indexOf(Object): Devuelve el índice el elemento especificado, de no encontrarlo devuelve -1
- remove(int): Elimina el elemento de la posición especificada
- set (int, Object): Reemplaza el elemento de la posición especificada en el primer arumento
- size(): Devuelve la cantidad de elementos de la lista

EJERCICIO DEL USO DE LAS CLASES LINKEDLIST Y ARRAYLIST

- Ver en clase

DEMO LINKEDLIST

- Ver ejercios en clase

EJERCICIO INTEGRADOR USANDO LISTAS

LISTA DE CONTACTOS PERSONALES:

Se requiere construir un programa que maneje una lista de contactos personales. Un contacto tiene nombre, apellidos, dirección, correo electrónico, teléfono y celular.

El programa debe poder:

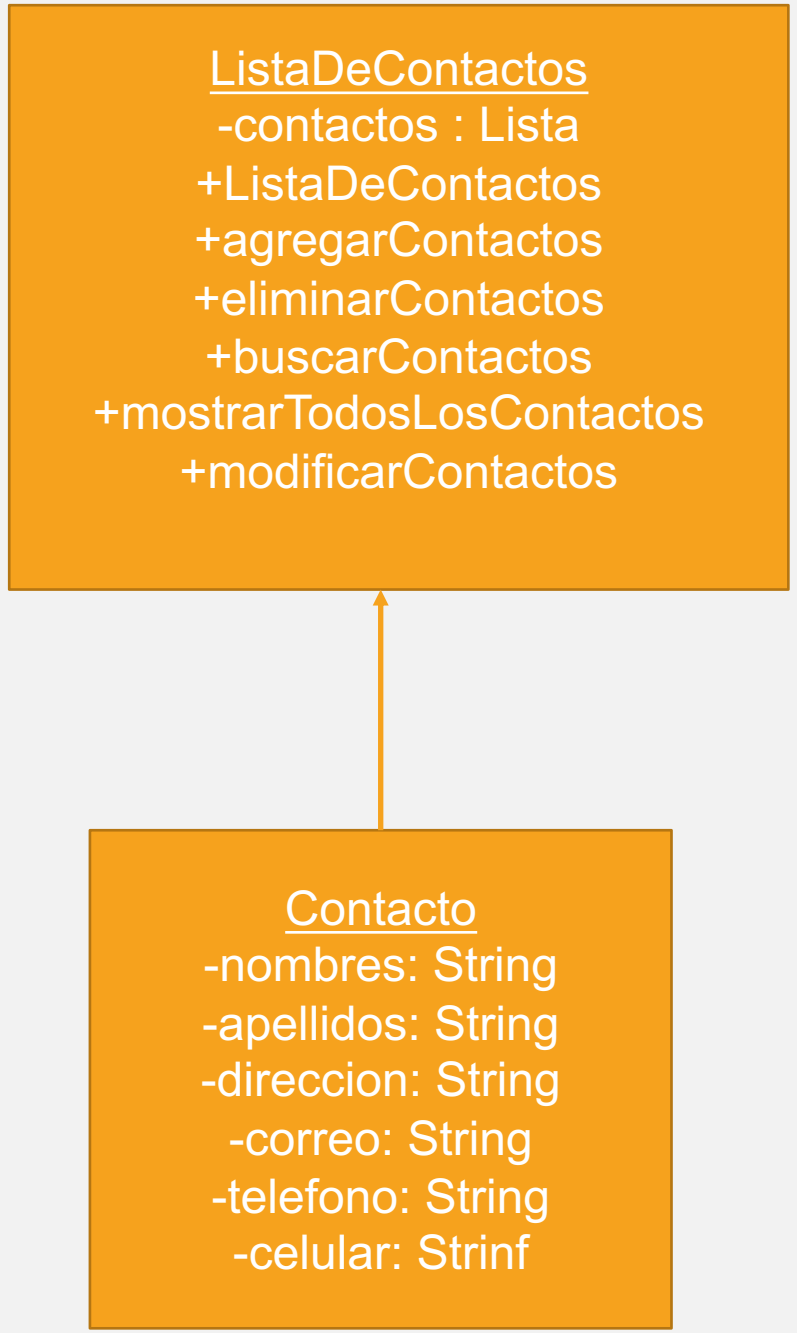
- Agregar un nuevo contacto
- Eliminar un contacto ya existente
- Ver la información detallada de un contacto
- Modificar datos de un contacto

Ver solución en clase

DIAGRAMA DE CLASES DEL PROBLEMA DE LISTA DE CONTACTOS PERSONALES

https://repl.it/join/rrsygyuv-lourdesarmenta

Cada contacto será un objeto tipo *Contacto*. Los diferentes contactos serán almacenados en unobjeto *ListaDeContactos*. Esta clase contiene métodos que permiten, agregar, buscar, ver, modificar, eliminar y listar contactos. En *ListaDeContactos* habrá un atributo tipo *Lista* (que almacenará todos lo contactos)

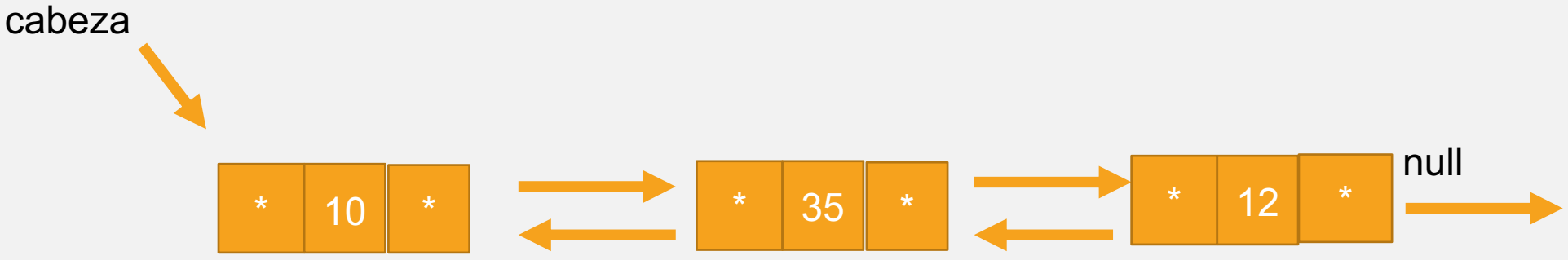


CODIGO FUENTE LISTA DE CONTACTOS

https://replit.com/@LourdesArmenta/listaContactos

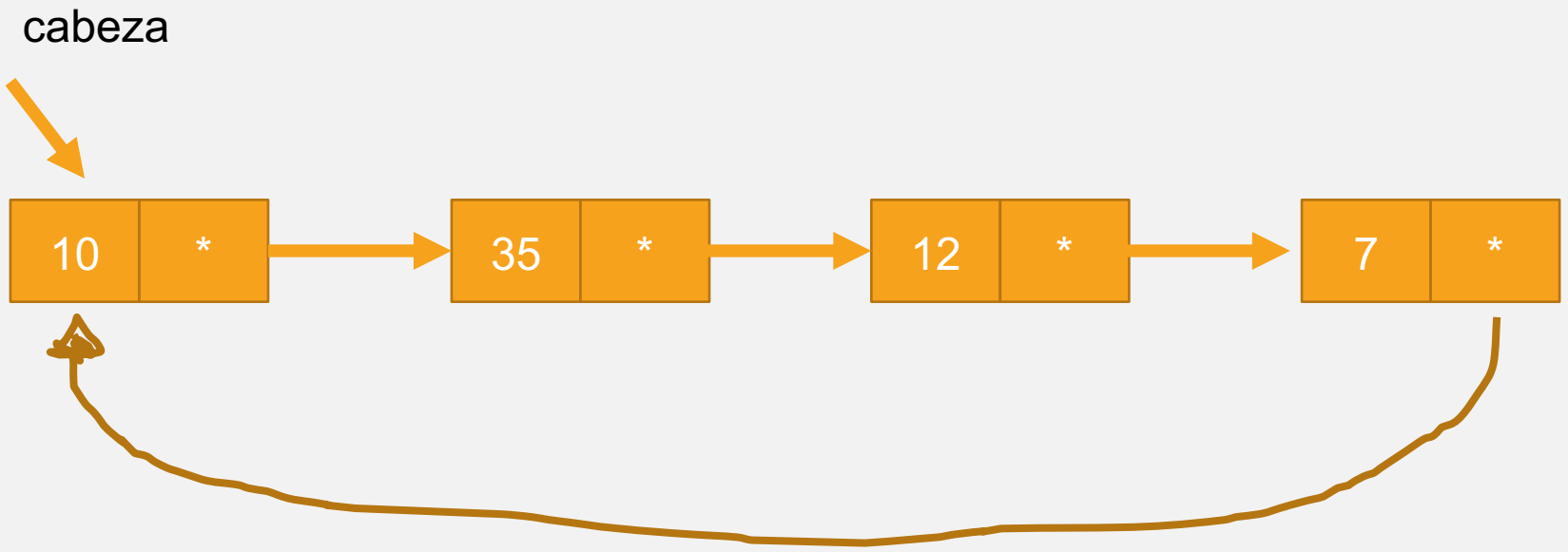
LISTAS DOBLEMENTE ENLAZADAS

- *Cada nodo tiene dos enlaces*: uno apunta al nodo anterior, o apunta al valor Nulo si el primer nodo; y otro que apunta al siguiente nodo, o apunta al valor Nulo si es el último nodo



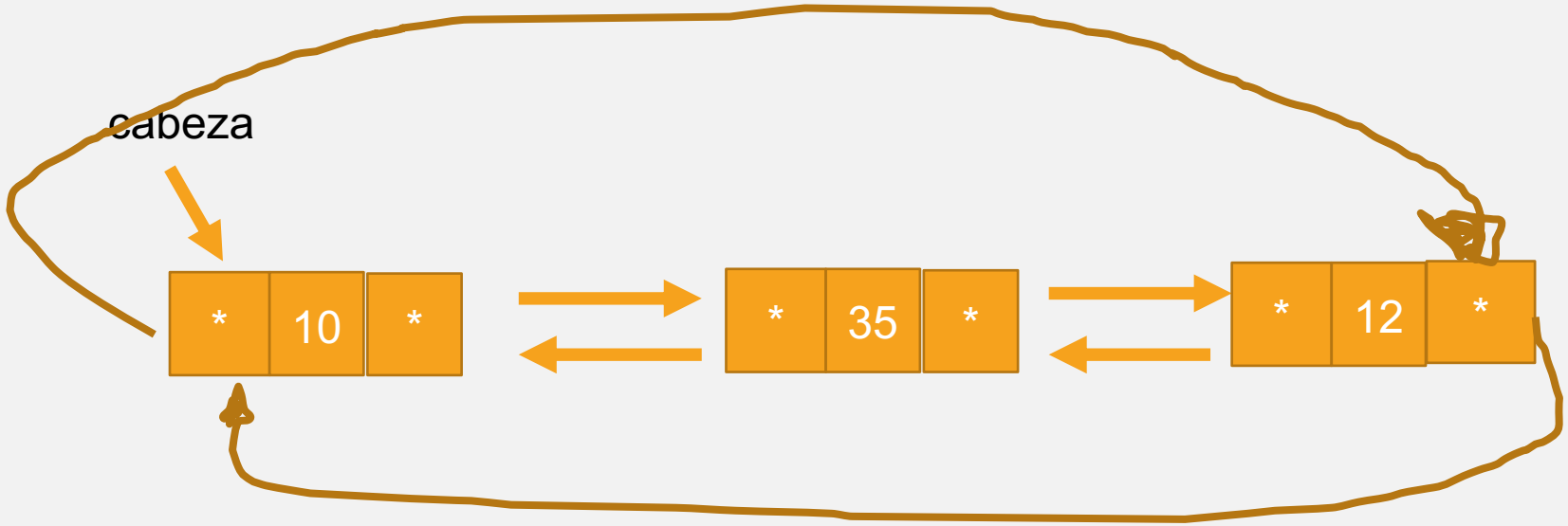
LISTAS CIRCULARES

- En una lista enlazada circular, el primer y el último nodo están unidos



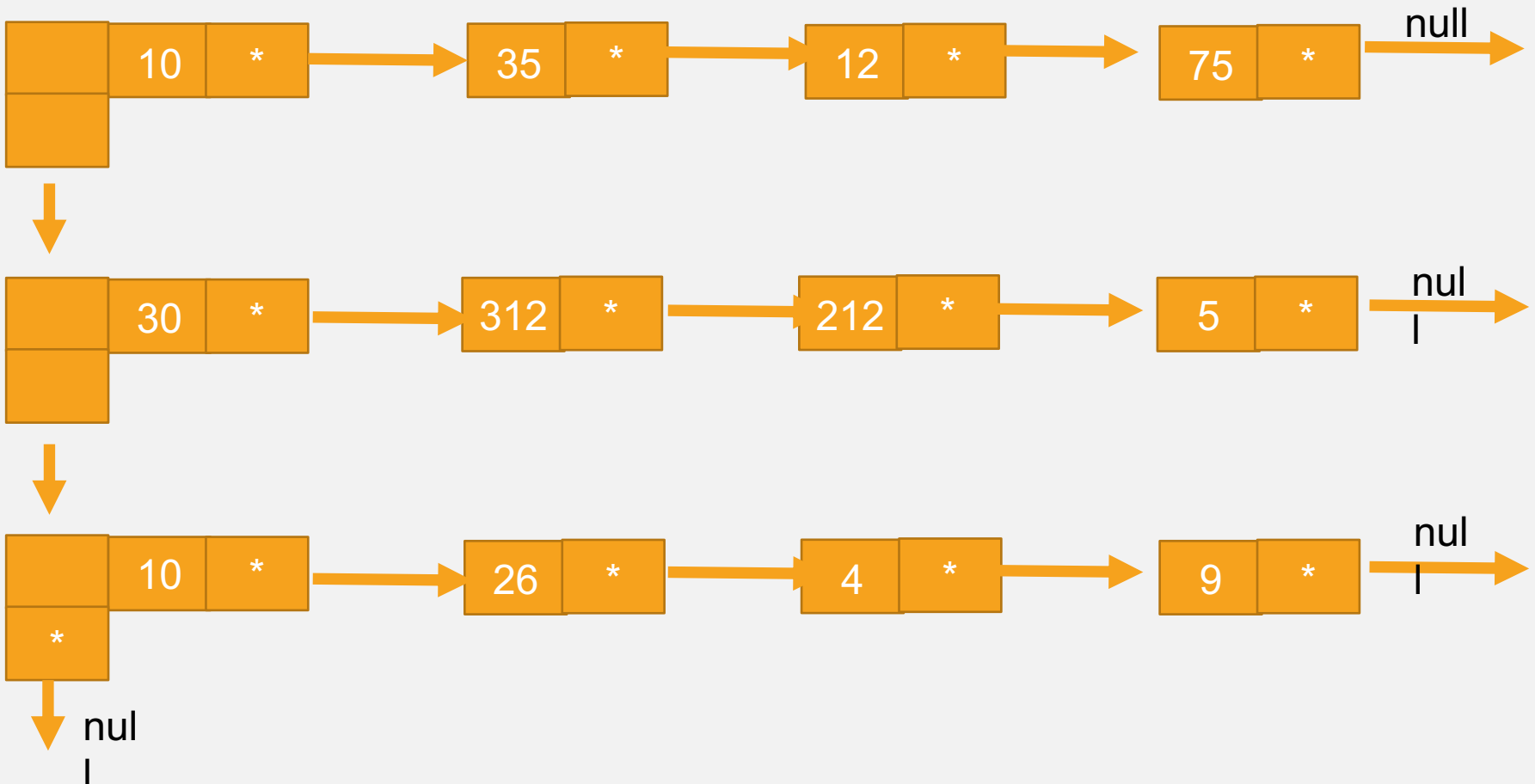
LISTAS DOBLEMENTE ENLAZADAS CIRCULARES

- Una lista circular puede también ser doblemente encadenada o enlazada



LISTA DE LISTAS

- En las listas de listas, el campo de datos de un nodo puede ser otra lista enlazada



IMPLEMENTACIÓN EN JAVA LISTA DOBLEMENTE LIGADA

- El Nodo típico es el mismo que para construir listas simples, salvo que tienen otra referencia al nodo anterior:

```
package listaDoblementeEnlazada;
public class Nodo <T>{
   //atributo valor de tipo T.Almacena la
   //referencia al objeto que se guarda
   //en el nodo
   private T valor;
   Nodo<T> siguiente;
   Nodo<T> anterior;
   //constructor por defecto
   public Nodo() {
      valor = null;
      siguiente = null;
      anterior = null;
}
public T getValor() {
   return valor;
   }
public void setValor(T valor) {
   this.valor = valor;
}
public Nodo<T> getSiguiente() {
   return siguiente;
}
public void setSiguiente(Nodo<T> siguiente) {
   this.siguiente = siguiente;
}
public Nodo<T> getAnterior() {
   return anterior;
}
   public void setAnterior(Nodo<T> anterior) {
   this.anterior = anterior;
   }
}
```

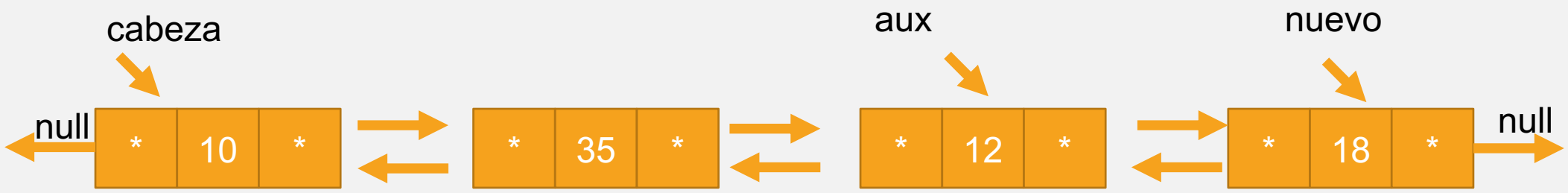
IMPLEMENTACIÓN EN JAVA LISTA DOBLEMENTE LIGADA

- El Nodo típico es el mismo que para construir listas simples, salvo que tienen otra referencia al nodo anterior:

```
package listaDoblementeEnlazada;
public class ListaDoblementeEnlazada<T> {
  //primer elemento de la lista
  private Nodo<T> cabeza;
  / /total de elementos de la lista
  private int tamanio;
  //constructor por defecto
  public ListaDoblementeEnlazada() {
  cabeza = null;
  tamanio =0;
  }
  /*
  * devuelve el tamaño de la lista
  */
  public int getTamanio() {
  return tamanio;
  }
  public boolean esVacia() {
  return (cabeza==null);
  }.
  //Agrega un nuevo nodo al final de la lista
  public void agregar(T valor) {
  }
 /. /inserta un nuevo nodo en la lista
  public void insertar(T valor, int pos)
  {
  }
  //devuelve el valor de una determinada posicion
  public T getValor(int pos)
  {
  }
  //elimina un nodo de una determinada posicion
  public void remover(int pos)
  {
  }
  //elimina todos los nodos de la lista
  public void limpiar() {
  cabeza= null;
  tamanio = 0;
  }
}
```

CONSIDERACIONES EN LA IMPLEMENTACIÓN

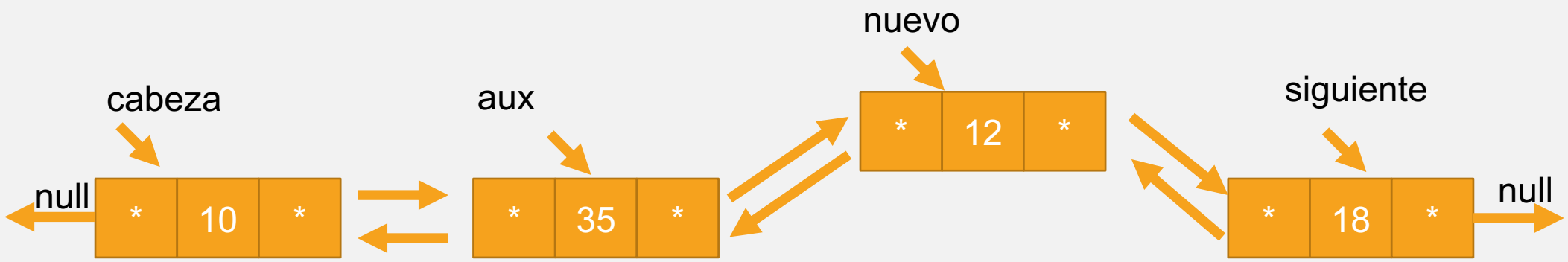
- Agregar: El funcionamiento de esta operaciópn es casi idéntica a la lista simple ligada. La única diferencia es que el nuevo nodo tiene el nuevo atributo anterior que debe apunta al nodo anterior




```
aux.setSiguiente(nuevo)
nuevo.setAnterior(aux);
```

CONSIDERACIONES EN LA IMPLEMENTACIÓN

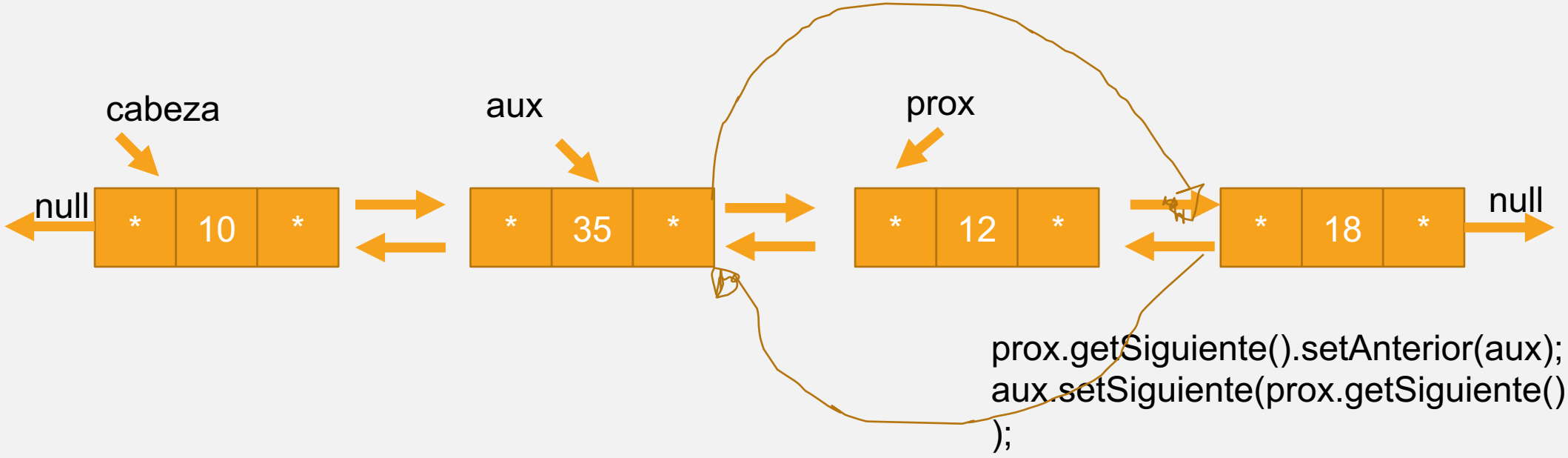
- insertar: Al igual que la opeación agregar, el insertar es casi idéntico a la lista enlazada simple, exepto lo que se ilustra




```
aux.setSiguiente(nuevo)
nuevo.setSiguiente(siguiente);
nuevo.setAnterior(aux);
siguiente.setAnterior(nuevo)
```

CONSIDERACIONES EN LA IMPLEMENTACIÓN

- **eliminar**: La operación remover es casi idéntico a la lista enlazada simple, se ilustra.



LISTA CIRCULAR

- **Para recorrer sobre una lista circular, se crea un nodo especial que hace las funciones del nodo cabecera y debido a que el atributo siguiente del último nodo no apunta a null, se utiliza el siguiente recorrido:**

https://replit.com/@LourdesArmenta/listaCircular

LISTA DOBLEMENTE CIRCULAR

- Son listas doblemente enlazadas, pero además son circulares, es decir, el último nodo apunta al primero, y el primero apunta al último
- El nodo que se utiliza en las listas doblemente enlazadas circulares es el mismo que de la lista doblemente enlazada, es decir cuenta con un atributo *valor*, un atributo *siguiente* que apunta al siguiente nodo y un atributo *anterior* que apunta al siguiente anterior.

https://replit.com/@LourdesArmenta/ListaDoblementeCircular

LISTAS ENLAZADAS VERSUS VECTORES O MATRICES

	VECTOR	LISTA ENLAZADA
INDEXADO	O(1)	O(n)
INSERCIÓN/ELIMINACIÓN AL FINAL	O(1)	O(1) ó $O(n)^2$
INSERCIÓN/ELIMINACIÓN A LA MITAD	O(n)	O(1)

1. Los elementos se puden insertar en una lista indefinidamente, en un vector tarde o temprano se llenará y requerirá ser redimensionado.
2. Los vectores permiten acceso aleatorio mientras que las listas solo permite el acceso secuencial.
3. Las listas simples solo pueden ser recorridas en una dirección, por lo que son inadecuadas para buscar un elemento por su índice rápidamente.
4. El acceso secuencial en los vectores es más rápido que las listas enlazadas.
5. Las listas requieren un almacenamiento extra para las referencias o ligas, por lo que en pequeños datos como caracteres o boleanos son impráctias
6. Las listas doblemente enlazadas requieren mas espacio por nodo por lo que las operaciones son mas costosas pero tienen mayor facilidad para manilupular.
7. Las listas circulares tiene la ventaja de recorrer la lista desde cualquier punto y permiten el acceso rápido al primer y último elemento