TADs Pilas y Cola, Aplicaciones

Estructura de Datos/Algoritmos I

Depto. de computación-UNRC
Prof. Pablo Castro

Implementación de TADs en Lenguajes OO

Podemos definir de una forma elegante los TADs en OO:

- El uso de interfaces o clases abstractas para definir TADs.

- El uso de herencia para proveer varias implementaciones de TADs.

- El uso de herencia para proveer estructuras polimórficas.

Esto minimiza la cantidad de cambios necesarios que deben hacerse cuando se cambian implementaciones.

TAD Pila

- ***Elementos:*** Secuencias de elementos de un tipo dado, en donde un extremo de la secuencia se la denomina *tope*.

- *Operaciones:*
  - *esVacia*: dice si una pila es vacía.
  - *vaciar*: se vacía la pila.
  - *longitud*: retorna la cantidad de elementos de la pila
  - *apilar*: apila un elemento en la pila.
  - *desapilar*: si la pila no es vacía, elimina el elemento en el tope.

  - *tope*: si la pila es no vacía, retorna el tope de la lista sin modificarla

Interface Pila

```
/* ------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------ */
/* Interface Pila: Interface que especifica el TAD Pila                     */
/* (polimorfico).                                                           */
/* Las clases que implementan el TAD Pila deben implementar esta, y         */
/* proveer implementaciones para todos los metodos abstractos de la         */
/* misma.                                                                   */
/* ------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------ */
public interface Pila {

    /* esVacia(): Indica si la pila es vacia o no                           */
    /* pre: true                                                            */
    /* post: Retorna true ssi la pila no tiene elementos                    */
    public abstract boolean esVacia();

    /* vaciar(): Elimina todos los elementos de la pila                     */
    /* pre: true                                                            */
    /* post: elimina todos los elementos de la pila.                        */
    public abstract void vaciar();


    /* longitud(): Retorna la cantidad de elementos de la pila              */
    /* pre: true                                                            */
    /* post: retorna la cantidad de elementos de la pila.                   */
    public abstract int longitud();


    /* apilar(item): apila item al tope de la pila.                         */
    /* pre: true                                                            */
    /* post: apila item al tope de la pila.                                 */
    /* Si el apilado falla por algun motivo, lanza una excepcion            */
    /* ExcepcionPila.                                                       */
    public abstract void apilar(Object item) throws ExcepcionPila;
```

Interface Pila

```
    /* desapilar(): elimina el elemento que se encuentra en el    */
    /* tope de la pila.                                           */
    /* pre: longitud()>=1                                         */
    /* post: si la pila es no vacia, se elimina el elemento en el */
    /* tope de la misma.                                          */
    /* Si la pila esta vacia, lanza una excepcion ExcepcionPila.  */
    public abstract void desapilar() throws ExcepcionPila;

    /* tope(): retorna el elemento en el tope de una pila no      */
    /* vacia.                                                     */
    /* pre: longitud()>=1                                         */
    /* post: si la pila es no vacia, retorna el item en el tope de*/
    /* la misma                                                   */
    /* Si la pila esta vacia, lanza una excepcion ExcepcionPila.  */
    public abstract Object tope() throws ExcepcionPila;

} // fin de interface Pila
```

Implementación sobre Arreglos

```
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */
/*  Clase PilaSobreArreglos: Implementacion del TAD Pila, usando un       */
/*  arreglo de enteros con tamanho maximo MAX_LIST.                       */
/*  Esta clase implementa los metodos abstractos declarados en Pila.      */
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */

public class PilaSobreArreglos implements Pila {

    private static final int MAX_PILA = 100; // numero maximo de items en la
                                             // pila.
    private Object items[];                  // arreglo usado para almacenar
                                             // los elementos de la pila.
    private int tope;                        // entero que indica el indice
                                             // corriente del tope de la pila.
                                             // tope == -1 si la pila esta
                                             // vacia.

/*----------------------------------------------------------------------*/
/*  PilaSobreArreglos(): Constructor de la clase PilaSobreArreglos.     */
/*  Pre: true.                                                          */
/*  Post: Se crea un arreglo de objetos de tamanho MAX_LIST, y se       */
/*  inicializa tope en -1.                                              */
/*----------------------------------------------------------------------*/
public PilaSobreArreglos() {

    items = new Object[MAX_PILA];
    tope = -1;

} // fin de PilaSobreArreglos()
```

Pila sobre Arreglos

```
/*-------------------------------------------------------------------------*/
/* esVacia(): Retorna true ssi la pila esta vacia.                         */
/* Pre: true.                                                              */
/* Post: retorna true ssi tope == -1.                                      */
/*-------------------------------------------------------------------------*/
public boolean esVacia() {

    return (tope == -1);

    } // fin de esVacia()


    /*-------------------------------------------------------------------------*/
    /* longitud(): returna el numero de elementos en la pila                   */
    /* Pre: true                                                               */
    /* Post: retorna el valor tope+1.                                          */
    /*-------------------------------------------------------------------------*/
    public int longitud() {

        return tope+1;

    } // fin de longitud()
    ...
} // fin de clase PilaSobreArreglos
```

Pilas sobre Listas Enlazadas

Usaremos memoria dinámica para implementar las pilas:

```
/*-----------------------------------------------------------------*/
/*-----------------------------------------------------------------*/
/* Clase Nodo: Usada para construir listas enlazadas para          */
/* implementaciones basadas en referencias de listas,              */
/* pilas, colas, etc.                                              */
/*-----------------------------------------------------------------*/
/*-----------------------------------------------------------------*/

public class Nodo {

    private Object item;
    private Nodo siguiente;

    /*-------------------------------------------------------------*/
    /* Nodo(): Constructor de la clase No                          */
    /*Pre: true.                                                   */
    /*Post: item y siguiente se setean en null.                    */
    /*-------------------------------------------------------------*/
    public Nodo() {
        item = null;
        siguiente = null;
    }
```

El atributo item es una referencia al dato guardado en el nodo

El atributo siguiente es una referencia al próximo nodo.

Pilas sobre Listas Enlazadas

```
/*-------------------------------------------------------------*/
/* Nodo(nuevoItem): Constructor de la clase Nodo.              */
/*Pre: true.                                                   */
/*Post: item se setea con nuevoItem, y siguiente se setea en  */
/* null.                                                      */
/*-------------------------------------------------------------*/
public Nodo(Object nuevoItem) {
    item = nuevoItem;
    siguiente = null;
}
```

Métodos para acceder y modificar los valores de los nodos.

```
/*-------------------------------------------------------------*/
/* Nodo(nuevoItem, siguienteNodo): Constructor de la clase     */
/* Nodo.                                                       */
/*Pre: true.                                                   */
/*Post: item se setea con nuevoItem, y siguiente se setea con */
/* siguienteNodo.                                             */
/*-------------------------------------------------------------*/
public Nodo(Object nuevoItem, Nodo siguienteNodo) {
    item = nuevoItem;
    siguiente = siguienteNodo;
}
```

Pilas sobre Listas Enlazadas

```
/*-------------------------------------------------------------------*/
/* cambiarItem(nuevoItem): setea el atributo item del nodo.          */
/*Pre: true.                                                         */
/*Post: el atributo item se setea con nuevoItem.                     */
/*-------------------------------------------------------------------*/
public void cambiarItem(Object nuevoItem) {
    item = nuevoItem;
}

/*-------------------------------------------------------------------*/
/* cambiarSiguiente(siguienteNodo): setea el atributo                */
/* "siguiente" de un nodo.                                           */
/* Pre: true.                                                        */
/*Post: siguiente se setea con siguienteNodo.                        */
/*-------------------------------------------------------------------*/
public void cambiarSiguiente(Nodo siguienteNodo) {
    siguiente = siguienteNodo;
}

/*-------------------------------------------------------------------*/
/* obtenerItem(): retorna el item de un nodo.                        */
/*Pre: true.                                                         */
/*Post: item is retornado, y no se cambia el valor de los           */
/* atributos.                                                       */
/*-------------------------------------------------------------------*/
public Object obtenerItem() {
    return item;
}


/*-------------------------------------------------------------------*/
/* obtenerSiguiente(): retorna el atributo "siguiente" de un         */
/* Nodo.                                                             */
/*Pre: true.                                                         */
/*Post: siguiente es retornado, y no se cambia el valor de los      */
/* atributos.                                                       */
/*-------------------------------------------------------------------*/
public Nodo obtenerSiguiente() {
    return siguiente;
}
```

Más métodos

Clase: *PilasobreListasEnlazadas*

```
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */
/* Clase PilaSobreListasEnlazadas: Implementacion del TAD Pila, usando    */
/* una lista enlazada de objetos de tipo Nodo, donde el primero es el     */
/* nodo que contiene el tope de la pila.                                  */
/* Esta clase implementa los metodos abstractos declarados en Pila.       */
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */

public class PilaSobreListasEnlazadas implements Pila {


    private Nodo tope; // usado como la cabeza del a lista enlazada que
                       // almacena los elementos de la pila.

/*-------------------------------------------------------------------------*/
/* PilaSobreListasEnlazadas(): Constructor de la clase                     */
/*  PilaSobreListasEnlazadas.                                              */
/* Pre: true.                                                              */
/* Post: Se inicializa tope en null.                                       */
/*-------------------------------------------------------------------------*/
public PilaSobreListasEnlazadas() {

    tope = null;

} // fin de PilaSobreListasEnlazadas()

/*-------------------------------------------------------------------------*/
/* esVacia(): Retorna true ssi la pila esta vacia.                         */
/* Pre: true.                                                              */
/* Post: retorna true ssi tope == null.                                    */
/*-------------------------------------------------------------------------*/
public boolean esVacia() {

    return (tope == null);

} // fin de esVacia()
```

Pilas sobre Listas Enlazadas

```
/*-------------------------------------------------------------------------*/
/* longitud(): returna el numero de elementos en la pila                   */
/* Pre: true                                                               */
/* Post: retorna el numero de nodos ligados a tope.                        */
/*-------------------------------------------------------------------------*/
public int longitud() {

    int numItems = 0;
    Nodo corriente = tope;
    while (corriente != null) {
    // inv: numItems == numero de nodos desde tope a corriente (sin
    // incluir este)
        numItems = numItems + 1;
        corriente = corriente.obtenerSiguiente();
    } // fin while
    return (numItems);

 } // fin de longitud()

/*-------------------------------------------------------------------------*/
/* vaciar(): elimina todos los elementos de la pila.                       */
/* Pre: true.                                                              */
/* Post: tope se setea en null.                                            */
/*-------------------------------------------------------------------------*/
public void vaciar() {

    tope = null;

} // fin de vaciar()
```

Pilas sobre Listas Enlazadas

```
/*-------------------------------------------------------------------------*/
/* apilar(item): apila item en el tope de la pila.                         */
/* Pre: true                                                               */
/* Post: se crea un nuevo nodo, se almacena item en este y se enlaza       */
/* a la cabeza de tope.                                                    */
 /*-------------------------------------------------------------------------*/
public void apilar(Object item) {
        Nodo nuevoNodo = new Nodo();
        nuevoNodo.cambiarItem(item);
        nuevoNodo.cambiarSiguiente(tope);
        tope = nuevoNodo;
} // fin de apilar(item)

/*-------------------------------------------------------------------------*/
/* desapilar(): elimina el item en el tope de una pila no vacia.           */
/* Pre: longitud()>0                                                       */
/* Post: si la pila es no vacia, se decrementa tope.                       */
/* Si la pila esta vacia, se lanza una excepcion ExcepcionPila.            */
/*-------------------------------------------------------------------------*/
public void desapilar() throws ExcepcionPila {
    if (tope!=null) {
        tope = tope.obtenerSiguiente();
    }
    else {
        throw new
        ExcepcionPila("PilaSobreListasEnlazadas.desapilar: pila vacia.");
    }
} // fin de desapilar()
```

Pilas sobre Listas Enlazadas

```
/*-------------------------------------------------------------------*/
/* tope(): retorna el item en el tope de una pila no vacia.          */
/* Pre: longitud()>0                                                 */
/* Post: si la pila es no vacia, se retorna items[tope].             */
/* Si la pila esta vacia, se lanza una excepcion ExcepcionPila.      */
/*-------------------------------------------------------------------*/
public Object tope() throws ExcepcionPila {
    if (tope!=null) {
        return tope.obtenerItem();
        }
        else {
            throw new
            ExcepcionPila("PilaSobreListasEnlazadas.tope: pila vacia.");
        }
    } // fin de tope()

} // fin de clase PilaSobreListasEnlazadas
```

También podríamos utilizar las listas ya implementadas para implementar las pilas.

Ventajas y Desventajas

- Pilas sobre arreglos:
  - simple, fácil de implementar y mantener.
  - limitada por el tamaño del arreglo.
  - se puede desperdiciar espacio.
  - tope, apilar y desapilar son eficientes.

- Pilas sobre listas enlazadas:
  - más compleja de implementar y mantener.
  - no está limitada en espacio.
  - cada elemento necesita más espacio.
  - tope, apilar y desapilar son eficientes.

- Pilas sobre listas ya implementadas:
  - más fácil de implementar y mantener.
  - dependemos de la implementación de listas.

Pilas Genéricas

También podemos usar genericidad para obtener una versión genérica de las pilas:

```
public interface PilaGen<AnyType>{
    //apilar(): apila un item al tope de la pila
    //Pre: true
    //post: apila el item al tope de la pila
    void apilar(AnyType item) throws ExceptionPila;
    // desapilar(): desapila el elemento al tope de la lista.
    // Pre: la pila no es vacia
    // Post: desapila el elemento al tope.
    void desapilar() throws ExceptionPila;
    // tope(): retorna el elemento al tope de la pila
    // Pre: la pila no es vacia
    // Post: devuelve el elemento al tope de la pila
    AnyType tope() throws ExceptionPila;
    // esVacia(): dice si la pila es vacia o no
    // Pre: true
    // Post: devuelve true ssi es vacia
    boolean esVacia();
    // vaciar(): vacia la pila de elementos
    // Pre: true
    // Post: deja la pila vacia
    void vaciar();
}
```

Ejercicio:
Implementar la pilas genéricas con listas enlazadas.

Aplicaciones de las Pilas

Las pilas tienen diversas aplicaciones en computación, algunos ejemplos:

- Las pilas son utilizadas para implementar la llamada a rutinas en la mayoría de los lenguajes de programación (eso hace posible la recursión, por ejemplo).

- Chequear el balanceo de símbolos (paréntesis, etc) se puede hacer fácilmente utilizando una pila.

- Los algoritmos para evaluar expresiones pueden ser implementados usando pilas.

Ejemplo de Recursión con Pilas

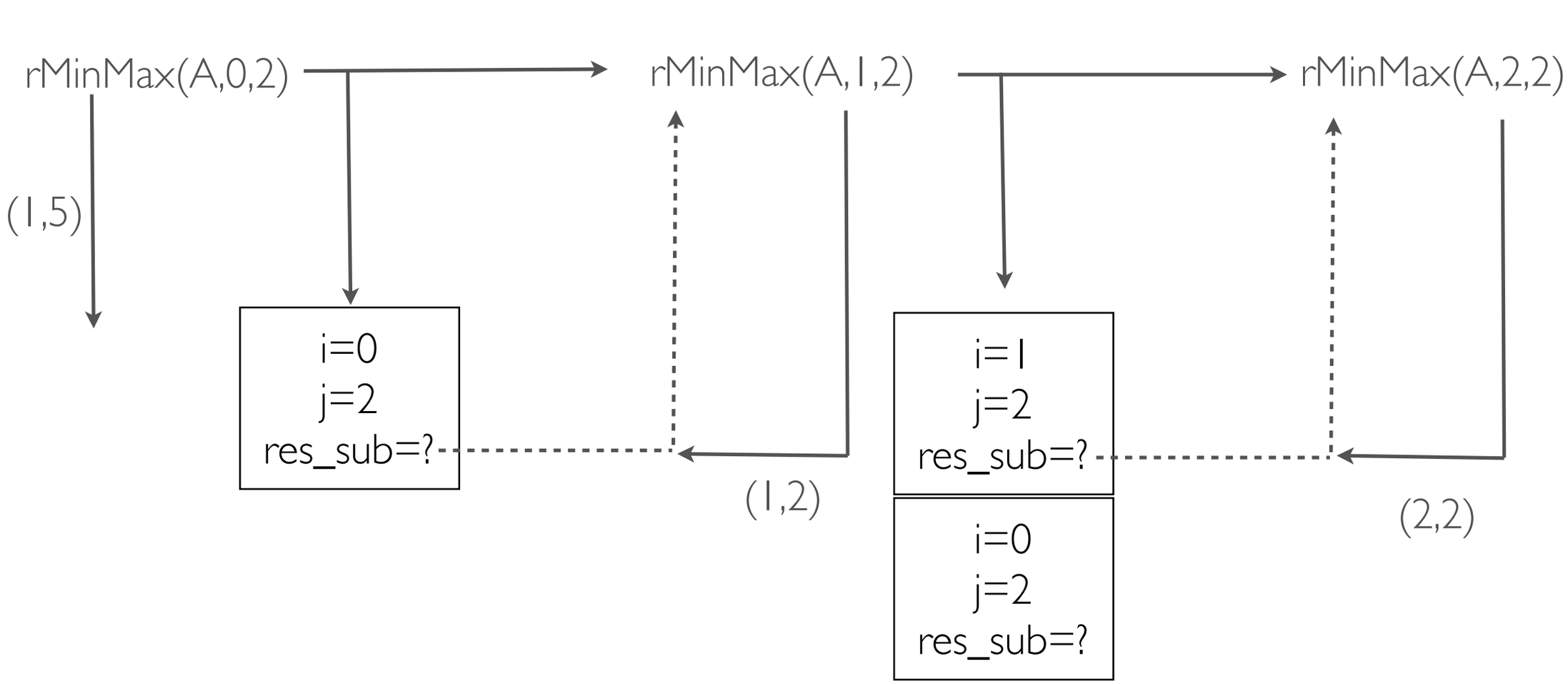
Veamos el ejemplo del rMinMax:

```
public static Pair rMinMaxArray(int[] A, int i, int j){
    if (i > j){
        Pair res = new Pair(0,0);
        return res; // special case: we return (0,0) if the array is empty
    }
    else{
    if (i==j){
        Pair res = new Pair(A[i],A[i]);
        return res; // base case: the array under consideration has size 1
}
else{
        Pair res_sub = new Pair();
        Pair res = new Pair();
        res_sub = rMinMaxArray(A, i+1, j);
        if (res_sub.fst() > A[i]){
            res.changeFst(A[i]);
        }
        else{
            res.changeFst(res\_sub.fst()); \\
        }
        if (res_sub.snd() <A[i]){
            res.changeSnd(A[i]);
        }
        else{
            res.changeSnd(res\_sub.snd()); \\
        }
        return res;
```

Java implementa la recursión utilizando una pila con registros de activación, los cuales guardan una "foto" del estado de las variables durante cada llamada recursiva

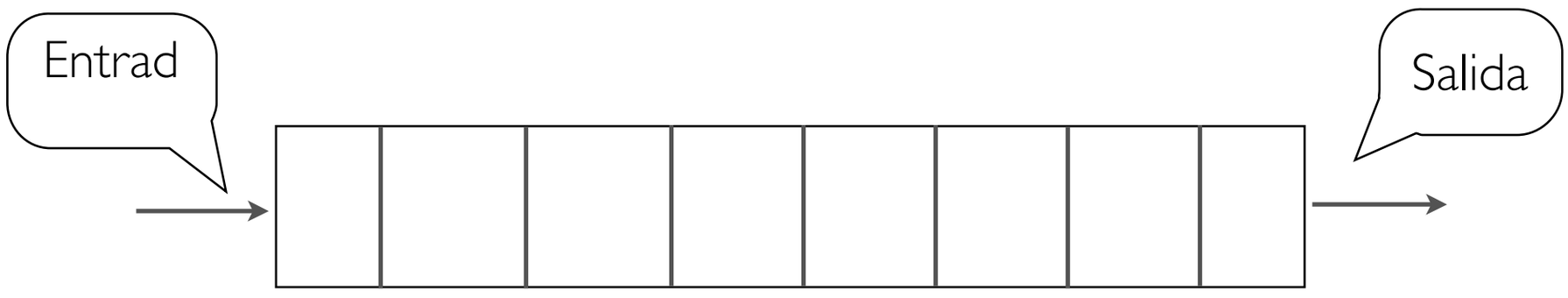
Recursión con Pilas

Supongamos A=[5,1,2]:



El TAD Cola

- Elementos: una sucesión finita de elementos de un tipo T. Un extremo se conoce como entrada y el otro como salida.
- Operaciones:
  - esVacia: dice si la cola es vacía.
  - vaciar: vacía la cola.
  - encolar: mete un elemento en la entrada
  - desencolar: saca un elemento de la salida.
  - longitud: devuelve la longitud.



Interface Cola

```
/* ------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------ */
/* Interface Cola: Interface que especifica el TAD Cola (polimorfico).      */
/* Las clases que implementan el TAD Cola deben heredar de esta, y proveer  */
/* implementaciones para todos los metodos abstractos de la misma.          */
/* ------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------ */
public interface Cola {

    /* esVacia(): Indica si la cola es vacia o no                           */
    /* pre: true                                                            */
    /* post: Retorna true ssi la cola no tiene elementos                    */
    public abstract boolean esVacia();

    /* vaciar(): Elimina todos los elementos de la cola                     */
    /* pre: true                                                            */
    /* post: elimina todos los elementos de la cola.                        */
    public abstract void vaciar();


    /* longitud(): Retorna la cantidad de elementos de la cola              */
    /* pre: true                                                            */
    /* post: retorna la cantidad de elementos de la cola.                   */
    public abstract int longitud();


    /* encolar(item): agrega item en el extremo de entrada de la cola.      */
    /* pre: true                                                            */
    /* post: agrega item en el extremo de entrada de la cola.               */
    /* Si la operacion falla por algun motivo, lanza una excepcion          */
    /* ExcepcionCola.                                                       */
    public abstract void encolar(Object item) throws ExcepcionCola;
```

INTERFACE COLA

```
    /* desencolar(): elimina el primer elemento en el extremo de salida   */
    /* de la cola .                                                       */
    /* pre: longitud()>=1                                                 */
    /* post: si la cola es no vacia, se elimina el primer elemento en el  */
    /* extremo de salida de la misma.                                     */
    /* Si la cola esta vacia, lanza una excepcion ExcepcionCola.          */
    public abstract void desencolar() throws ExcepcionCola;

    /* primero(): retorna el primer elemento en el extremo de salida.     */
    /* de la cola .                                                       */
    /* pre: longitud()>=1                                                 */
    /* post: si la cola es no vacia, retorna el primer elemento en el     */
    /* extremo de salida de la misma.                                     */
    /* Si la cola esta vacia, lanza una excepcion ExcepcionCola.          */
    public abstract Object primero() throws ExcepcionCola;

} // fin de interface Cola
```

Implementación de Colas con Arreglos

Una implementación simple seria la siguiente:

```
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */
/* Clase ColaSobreArreglosV1: Implementacion del TAD Cola, usando un      */
/* arreglo de enteros con tamanho maximo MAX_COLA, y cursores para los dos */
/* extremos.                                                              */
/* Esta clase implementa los metodos abstractos declarados en Cola.       */
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */

public class ColaSobreArreglosV1 implements Cola {

    private static final int MAX_COLA = 100; // numero maximo de items en la
                                             // cola.
    private Object items[];                  // arreglo usado para almacenar
                                             // los elementos de la cola.
    private int comienzo;                    // entero que indica el indice
                                             // correspondiente al primer
                                             // elemento de la cola.
    private int fin;                         // entero que indica el indice
                                             // que marca el fin de la cola.
                                             // La cola esta vacia si
                                             // fin < comienzo.
```

Colas con Arreglos

```
/*-------------------------------------------------------------------*/
/* ColaSobreArreglosV1(): Constructor de la clase ColaSobreArreglosV1.*/
/* Pre: true.                                                        */
/* Post: Se crea un arreglo de objetos de tamanho MAX_COLA, y se     */
/* inicializan comienzo en 0 y fin en -1.                            */
/*-------------------------------------------------------------------*/
public ColaSobreArreglosV1() {

    items = new Object[MAX_COLA];
    comienzo = 0;
    fin = -1;

} // fin de ColaSobreArreglosV1()
```

```
/*-------------------------------------------------------------------*/
/* esVacia(): Retorna true ssi la cola esta vacia.                   */
/* Pre: true.                                                        */
/* Post: retorna true ssi fin < comienzo.                            */
/*-------------------------------------------------------------------*/
public boolean esVacia() {

    return (fin < comienzo);

} // fin de esVacia()
```

```
/*-------------------------------------------------------------------*/
/* longitud(): returna el numero de elementos en la cola             */
/* Pre: true                                                         */
/* Post: retorna el valor tope+1.                                    */
/*-------------------------------------------------------------------*/
public int longitud() {

    if (fin < comienzo) {
        return (0);
    }
    else {
        return (fin-comienzo+1);
    }

} // fin de longitud()
```

Problemas de esta Implementación

- Sufre un corrimiento a la derecha: los items son empujados hacia el final, y el principio se puede acercar al Max_Cola -1

- Se desperdicia espacio.

- Una posible solución es mover los elementos al principio cuando se acercan al fin, pero esto es ineficiente.

Colas Circulares

```
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */
/* Clase ColaSobreArreglos: Implementacion del TAD Cola, usando un        */
/* arreglo de enteros con tamanho maximo MAX_COLA, y cursores para los dos */
/* extremos, viendo al arreglo como un ''arreglo circular''.              */
/* Esta clase implementa los metodos abstractos declarados en Cola.       */
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */

public class ColaSobreArreglos implements Cola {

  private static final int MAX_COLA = 100; // numero maximo de items en la
                                           // cola.
  private Object items[];                  // arreglo usado para almacenar
                                           // los elementos de la cola.
  private int comienzo;                    // entero que indica el indice
                                           // correspondiente al primer
                                           // elemento de la cola.
  private int fin;                         // entero que indica el indice
                                           // que marca el primer lugar.
                                           // libre en el arreglo.
                                           // La cola esta vacia si
                                           // fin == comienzo.
```

Se usan dos indices para indicar el primer elemento, y el primer lugar vacío, se incrementan utilizando aritmética mod MAX_COLA

Colas sobre Arreglos Circulares

```
/*-------------------------------------------------------------------*/
/* ColaSobreArreglos(): Constructor de la clase ColaSobreArreglos.   */
/* Pre: true.                                                        */
/* Post: Se crea un arreglo de objetos de tamanho MAX_COLA, y se     */
/* inicializan comienzo en 0 y fin en 0.                             */
/*-------------------------------------------------------------------*/
public ColaSobreArreglos() {

    items = new Object[MAX_COLA];
    comienzo = 0;
    fin = 0;

} // fin de ColaSobreArreglos()

/*-------------------------------------------------------------------*/
/* esVacia(): Retorna true ssi la cola esta vacia.                   */
/* Pre: true.                                                        */
/* Post: retorna true ssi fin == comienzo.                           */
/*-------------------------------------------------------------------*/
public boolean esVacia() {

    return (fin == comienzo);

} // fin de esVacia()
...
```

Ejercicio: implementar las otras operaciones

Colas con Listas Enlazadas

Una forma de simplificar la implementación es por medio de listas circulares:

```
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */
/* Clase ColaSobreListasEnlazadas: Implementacion del TAD Cola, usando    */
/* listas enlazadas, de manera circular, es decir, el ultimo nodo de la   */
/* lista enlazada se enlaza al primero de la lista.                       */
/* Esta clase implementa los metodos abstractos declarados en Cola.       */
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */

public class ColaSobreListasEnlazadas implements Cola {

    private Nodo fin;                       // referencia al ultimo nodo
                                            // de la lista enlazada circular.
                                            // La lista enlaza esta vacia si
                                            // fin == null.

    /*--------------------------------------------------------------------*/
    /* ColaSobreListasEnlazadas(): Constructor de la clase                */
    /* ColaSobreListasEnlazadas.                                          */
    /* Pre: true.                                                         */
    /* Post: Se inicializa fin en null.                                   */
    /*--------------------------------------------------------------------*/
    public ColaSobreListasEnlazadas() {

        fin = null;

    } // fin de ColaSobreListasEnlazadas()
```

Solo tenemos una referencia al fin de la cola.

Colas con Listas Circulares

```
/*-------------------------------------------------------------------------*/
/* esVacia(): Retorna true ssi la cola esta vacia.                         */
/* Pre: true.                                                              */
/* Post: retorna true ssi fin == null.                                     */
/*-------------------------------------------------------------------------*/
public boolean esVacia() {

    return (fin == null);

} // fin de esVacia()
```

```
/*-------------------------------------------------------------------------*/
/* encolar(item): agrega item en el extremo de entrada de la cola.         */
/* pre: true                                                               */
/* post: agrega item en el extremo de entrada de la cola, es decir, al     */
/* final de la misma.                                                      */
/* Si la operacion falla por algun motivo, lanza una excepcion             */
/* ExcepcionCola.                                                          */
/*-------------------------------------------------------------------------*/
public void encolar(Object nuevoItem) throws ExcepcionCola {
    Nodo nuevoNodo = new Nodo(nuevoItem);
    if (esVacia()) {
        nuevoNodo.cambiarSiguiente(nuevoNodo);
    }
    else {
        nuevoNodo.cambiarSiguiente(fin.obtenerSiguiente());
        fin.cambiarSiguiente(nuevoNodo);
    }
    fin = nuevoNodo;
} // fin de encolar(item)
```

Se pone al elemento a encolar como siguiente del fin

Ventajas y Desventajas

- Implementadas sobre arreglos:
  - La implementación es simple y elegante.
  - Se tienen restricciones de memoria.
  - Las operaciones son eficientes.

- Implementadas sobre listas circulares:
  - Simple y fácil de mantener.
  - Cada elemento ocupa más espacio.
  - No hay restricciones de memoria.

Aplicaciones del TAD cola

El TAD cola es muy útil en computación:

- Comunicación asincrónica: cuando dos programa/procesos se necesitan comunicar.

- Colas de impresión!

- Cuando tenemos un recurso compartido, el acceso al recurso puede ser organizado utilizando colas.

- En simulación, la noción de cola es fundamental para simular procesos.

- Podemos usar una cola y una pila para reconocer palíndromos.