Estructura de Datos
Grado en Ingeniería Informática en Sistemas de Información - Curso 2012/2013
Enseñanzas de Prácticas y Desarrollo

EPD-7: Pilas y Colas

Objetivos

- Introducir los conceptos de pilas y colas
- Dominar el uso de las estructuras de pilas y colas en Java
- Realizar aplicaciones basadas en las estructuras de pilas y colas.

Conceptos

1. El Concepto de Pila

Una pila (stack) es un tipo especial de lista lineal en la que la inserción y borrado de nuevos elementos se realiza sólo por un extremo denominado cima o tope (*top*).
La pila es una estructura con numerosas analogías a la vida real: una pila de platos, una pila de monedas, una pila de cajas de zapatos, una pila de bandejas, etc.
Dado que las operaciones de insertar y eliminar se realizan por un solo extremo (el superior), los elementos sólo pueden eliminarse en orden inverso al que se insertan en la pila. En otras palabras, se utiliza un esquema LIFO (last-in, first-out. Último en entrar, primero en salir).
Las operaciones principales asociadas a las pilas son:

- *push*: operación de insertar un elemento en la pila
- *pop*: operación de eliminar un elemento de la pila
- *size*: calcula cuantos elementos hay en la pila
- *top*: devuelve sin borrar el primer elemento de la pila
- *isEmpty*: indica si la pila está vacía

En clase de teoría se ha visto como implementar una pila utilizando un array. Se ha visto también que esta implementación tiene un defecto: la pila tiene una capacidad limitada.
Para obviar a esta limitación, se puede utilizar una lista enlazada, debiendo implementar una clase auxiliar que modela el nodo de la lista que se va a utilizar. La estructura de esta clase sería la siguiente:

```
clase ElementoPila {
        Object elemento;
        ElementoPila next;
}
```

De esta forma, los nodos de lista enlazada, se enlazan unos con otros usando el atributo de la clase llamado *next*. Por otro lado, cada nodo, por separado, utiliza el atributo *elemento* para apuntar al objeto que hay que guardar, como se aprecia en la figura 1.

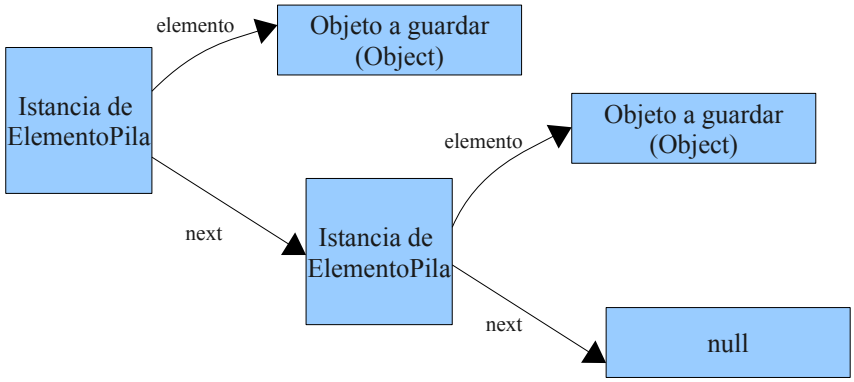

Figura 1: Estructura de lista enlazada.

En la figura se muestra una pila de dos elementos. Nótese que el atributo next del último elemento de una pila apunta a null.
De esta forma, la pila tendría la siguiente estructura:

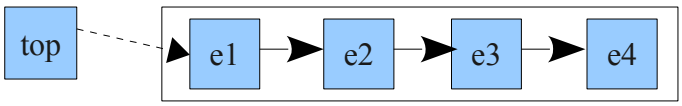

Figura 2: Ejemplo de pila implementada con una lista enlazada.

La implementación tendrá que tener por lo menos un atributo, *top* de tipo *ElementoPila*, que apunta al primer elemento de la pila.

Hay que subrayar que la pila debe permitir almacenar cualquier tipo de objeto, e implementar por lo menos los métodos mencionados anteriormente.

2. El Concepto de Cola

Una cola (queue) es una estructura de datos similar a la pila, pero donde la inserción y eliminación de los elementos se efectúa según el principio FIFO (First In First Out). Es decir, el primer elemento en salir es el primero que entró en la cola.

Como en el caso de la pila, la implementación basada en array de una cola tiene el defecto de establecer el número máximo de elementos que la estructura puede contener. En esta práctica, implementaremos una cola con una lista enlazada.
Como en el caso de la pila, la estructura que definiremos tendrá un atributo *top*, que apunta a la cabeza de la cola, que será el elemento más antiguo. En el caso de la cola, tendremos que mantener otro atributo *tail* que apunte al otro extremo de la cola, es decir que representa la última posición de la cola (el siguiente nodo de tail es null).
Por tanto, la extracción de la cola consiste en devolver el objeto asociado al nodo de la cabeza y desenlazarlo. Por el contrario, la inserción de un nuevo objeto consiste en localizar el último nodo y enlazar a continuación el nuevo nodo con dicho objeto. Para evitar tener que recorrer la lista siempre para encontrar el último nodo, se mantiene el atributo tail que lo referencia.
Como en el caso de la pila, los nodos en una lista enlazada, se enlazan unos con otros usando un atributo next. Con tal fin podemos definir una clase similar a la clase *ElementoPila.*
De esta forma, la estructura de la cola se establecerá por medio de dos atributos; uno top, que apunta a la cabeza de la cola, por donde se extraen los elementos; y otro, tail, por donde se introducen los elementos. De esta manera, se cumple la estructura FIFO.

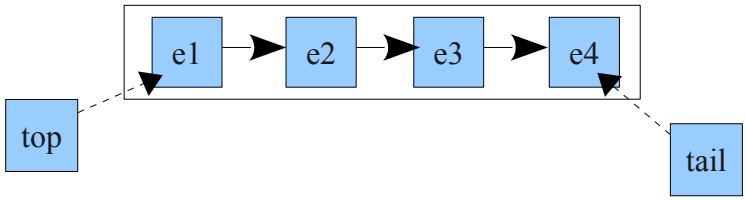

Figura 3: Ejemplo de cola implementada con una lista enlazada.

En la figura 3 se puede apreciar la estructura que tendrá una cola implementada con una lista enlazada. Los atributos top y tail apuntan a los extremos de la cola.

Bibliografía Básica

Fundamentos de programación: Algoritmos, Estructuras de datos y objetos. L. Joyanes. MacGraw-Hill, 2003, pag. 465-482.

Experimentos

E1. Analice el siguiente fragmento de código correspondiente a la clase ElementoPila:

```
public class NodoPila  {
       Object elemento;
       NodoPila next;
       public NodoPila(Object o) {
          elemento= o;
          next= null;
       }
            //OTROS MÉTODOS
   }
```

¿Por qué el tipo del atributo elemento es Object? ¿Qué finalidad tiene el atributo next?

E2. Analice el siguiente código correspondiente a la interfaz Pila:

```
public interface Pila {
   boolean isEmpty();
   Object pop() throws PilaVaciaException;
   void push(Object o);
   int size();
   Object top() throws PilaVaciaException;
}
```

¿Por qué se pueden lanzar las dos excepciones? ¿Por qué, a contrario de lo que se ha visto en clase de teoría, el método push no lanza ninguna excepción?

E3. Implemente la clase NodoCola, que se utilizará para los elementos de una cola. La clase tendrá que implementar la siguiente interfaz:

```
public interface QueueNode {
   Object getElem();
   NodoCola getNext();
   void setElem(Object elem);
   void setNext(NodoCola next);
}
```

E4. Analice el siguiente código correspondiente a la interfaz Queue:

```
public interface Queue {
   int buscar(Object o);
   Object dequeue() throws EmptyQueueException;
   void enqueue(Object o);
   Object front() throws EmptyQueueException;
   boolean isEmpty();
   int size();
}
```

¿Por qué se pueden lanzar las dos excepciones? ¿Cómo se podría implementar el método buscar?

Ejercicios

EJ1. Defina la excepción PilaVaciaException.

EJ2. Implemente la clase PilaEnlazada que implemente la interfaz vista en el experimento 2. La pila tendrá que ser implementada por medio de una lista enlazada.

EJ3. Cree un programa que demuestre el funcionamiento de los elementos desarrollados en los ejercicios 1 y 2.

EJ4. Defina la excepción EmptyQueueException.

EJ5. Implemente la clase LinkedQueue que implemente la interfaz vista en el experimento 4. La cola tendrá que ser implementada por medio de una lista enlazada.

EJ6. Cree un programa que demuestre el funcionamiento de los elementos desarrollados en los ejercicios 4 y 5.

EJ7. Implemente una clase LinkedList que implemente una lista enlazada. La clase tendrá que implementar la siguiente interfaz:

```
public interface ILinkedList {
   //añade un elemento en posición index, desplaza los siguientes
   void add(Object newElement, int index) throws IndexOutOfBoundsException;
   //añade un elemento al final
   void addEnd(Object newElement);
   //añade un elemento al principio
   void addFront(Object newElement);
   //devuelve el elemento en posición index
   Object getElement(int index) throws IndexOutOfBoundsException;
   boolean isEmpty();
   //borra elemento en posición index
   void remove(int index) throws IndexOutOfBoundsException;
   //borra último elemento
   void removeEnd();
   //borra primer elemento
   void removeFront();
   int size();
   String toString();
}
```

Cree un programa que demuestre el funcionamiento de la clase.

Problemas

P1. Implementar la clase Pila que implemente la siguiente interfaz:

```
public interface IStack {
   public int size();
   public boolean isEmpty();
   public Object top()  throws PilaVaciaException;
   public void push(Object element) throws PilaLlenaException;
   public Object pop()  throws PilaVaciaException;
}
```

La pila se implementará utilizando un array de tamaño 1000.
La excepción PilaVaciaException es la misma que la implementada en el ejercicio 1, la excepción PilaLlenaException deberá ser implementada. Esta última excepción deberá ser lanzada al intentar un elemento en una pila llena.
Cree un programa que demuestre el funcionamiento de la clase implementada.

P2. Implementar la clase Cola, que implemente la interfaz del ejercicio 4 utilizando un array de tamaño 1000. Como en el problema precedente, para el array se puede utilizar la clase Vector. En este caso, como en el problema 1, si se intenta insertar un elemento en una cola llena, se tendrá que lanzar una excepción.
Cree un programa que demuestre el funcionamiento de la clase implementada.

P3. Añade un método a la clase LinkedQueue que devuelva el objeto colocado en antepenúltimo lugar de la cola, sin extraerlo. Debe devolver null si la cola contiene dos o menos elementos. Comprueba que funcione correctamente.

P4. Añade un método a la clase LinkedQueue que reciba como parámetro un objeto y devuelva un entero indicando su posición en la cola (-1 si no está, 1 el primero, 2 el segundo, etc.) ¿Qué diferencia existe entre compararlos con equals o utilizando las referencias? Comprueba que funcione correctamente.

P5. Añade un método a la clase LinkedQueue que reciba como parámetro una referencia a un objeto QueueNode y un entero (void insertar(QueueNode sublista, int posicion)). El método debe insertar la lista que comienza con sublista justo a continuación del nodo cuya posición indique posicion. Por ejemplo, si en la lista [a-b-c-d-e] se desea insertar la sublista [x,y,z] en la posición 3, la lista resultante será [a-b-c-x-y-z-d-e]. Se seguirá el siguiente convenio:

1. Si posicion <=0, inserta la sublista justo al principio.
2. Si posicion >= num. de elementos, inserta la sublista justo al final.

Comprueba que funcione correctamente.

Ampliación de Bibliografía

- Arrays http://download.oracle.com/javase/6/docs/api/java/util/Arrays.html
- Vector http://download.oracle.com/javase/1.4.2/docs/api/java/util/Vector.html
- http://www.cs.cmu.edu/~adamchik/15-121/lectures/Stacks%20and%20Queues/Stacks%20and%20Queues.html
- BufferedReader http://download.oracle.com/javase/1.4.2/docs/api/java/io/BufferedReader.html
- BufferedReader http://www.java2s.com/Tutorial/Java/0180__File/CreateBufferedReaderfromFileReader.htm

Datos de la Práctica

Estimación temporal:

- Parte presencial: 120 minutos.
  - Explicación inicial: 20 minutos.
  - Experimentación: 35 minutos.
    - Experimento 1: 5 minutos.
    - Experimento 2: 15 minutos.
    - Experimento 3: 15 minutos.
  - Ejercicios: 65 minutos.
- Parte no presencial: 360 minutos.
  - Lectura y estudio del guión y bibliografía básica: 120 minutos
  - Problemas: 240 minutos.