Tema 1 - Ejercicios Resueltos

Estructuras de datos y algoritmos (Universitat Politecnica de Valencia)

TEMA 1

Estructuras de datos en Java

EJERCICIOS RESUELTOS

Ejercicio 1.- Implementa las clases *ArrayPila* y *LEGPila*, implementaciones del modelo *Pila* mediante un *array* y una lista enlazada, respectivamente.

Solución:

```
public class ArrayPila<E> implements Pila<E> {
  // Atributos
  protected E elArray[];
  protected int tope;
  protected static final int CAPACIDAD_POR_DEFECTO = 50;

  // Constructor
  @SuppressWarnings("unchecked")
  public ArrayPila() {
    elArray = (E []) new Object[CAPACIDAD_POR_DEFECTO];
    tope = -1;
  }

  // Inserta x en el tope de la Pila
  public void apilar (E x) {
    if (tope + 1 == elArray.length) duplicarPila();
    tope++;
    elArray[tope] = x;
  }

  // SII !esVacia(): elimina de la Pila el dato que ocupa su tope y lo devuelve
  public E desapilar () {
    E elUltimo = elArray[tope];
    tope--;
    return elUltimo;
  }

  // SII !esVacia(): obtiene el dato que ocupa el tope de la pila
  public E tope() { return elArray[tope]; }

  // Comprueba si la pila está o no vacía
  public boolean esVacia() { return (tope == -1); }

  // Duplicamos la capacidad del array
  @SuppressWarnings("unchecked")
  private void duplicarPila() {
    E nuevo[] = (E []) new Object[elArray.length*2];
    for (int i = 0; i <= tope; i++) nuevo[i] = elArray[i];
    elArray = nuevo;
  }
}
```

```
public class LEGPila<E> implements Pila<E> {
  // Atributos
  protected NodoLEG<E> tope;

  // Constructor
  public LEGPila() {
    tope = null;
  }

  // Inserta x en el tope de la Pila
  public void apilar(E x) {
    tope = new NodoLEG<E>(x, tope);
  }

  // SII !esVacia(): elimina el dato del tope de la Pila y lo devuelve
  public E desapilar() {
    E dato = tope.dato;
    tope = tope.siguiente;
    return dato;
  }

  // SII !esVacia(): obtiene el dato que ocupa el tope de la Pila
  public E tope() {
    return tope.dato;
  }

  // Comprueba si la Pila está o no vacía
  public boolean esVacia() {
    return (tope == null);
  }
}
```

Ejercicio 2.- Implementa la interfaz *Cola* mediante una *ListaConPI* (suponer que tenemos la clase *LEGListaConPI* como implementación de esta interfaz)

Solución:

```
public class LPICola<E> extends LEGListaConPI<E> implements Cola<E> {

  public void encolar(E e) {
    fin();
    insertar(e);
  }

  public E desencolar() {     // SII !esVacia()
    inicio();
    E primero = recuperar();
    eliminar();
    return primero;
  }

  public E primero() {  // SII !esVacia()
    inicio();
    return recuperar();
  }

}
```

Ejercicio 3.- Amplia la funcionalidad de la EDA *Lista con Punto de Interés* vía herencia con los siguientes métodos:

- *void* ***buscar****(E x)*: sitúa el PI sobre *x*. Si el dato no se encuentra se colocará el PI al final de la lista
- *void* ***vaciar****()*: vacía la lista
- *int* ***talla****()*: devuelve el número de elementos que contiene la lista
- *String* ***toString****()*: devuelve una cadena con la descripción de los elementos de la lista
- *void* ***invertir****()*: invierte el orden de los elementos de la lista
- *void* ***eliminar****(E x)*: elimina de la lista todos los elementos iguales a *x*.

Utiliza para ello únicamente los métodos existentes en el modelo *ListaConPI*.

Solución:

```
public interface ListaConPIExt<E> extends ListaConPI<E> {
  void buscar(E x);
  void vaciar();
  int talla();
  void invertir();
  void eliminar(E x);
}

public class LEGListaConPIExt<E> extends LEGListaConPI<E>
       implements ListaConPIExt<E> {

  public void buscar(E x) {
    inicio();
    while (!esFin() && !recuperar().equals(x)) siguiente();
  }

  public void vaciar() {
    inicio();
    while (!esVacia()) eliminar();
  }

  public int talla() {  // Si no estuviera el método talla() en ListaConPI
    int n = 0;
    for (inicio(); !esFin(); siguiente()) n++;
    return n;
  }

  public void invertir(){
    if (!esVacia()) {
      inicio();
      E dato = recuperar();
      eliminar();
      invertir();
      insertar(dato);
    }
  }

  public void eliminar(E x) {
    inicio();
    while (!esFin())
      if (recuperar().equals(x)) eliminar();
      else siguiente();
  }
```

```
  public String toString() {
    String res = "";
    for (inicio(); !esFin(); siguiente())
      res += recuperar().toString() + "\n";
    return res;
  }
}
```

Ejercicio 4.- Amplia la funcionalidad de la EDA *Pila* vía herencia para añadir un nuevo método que devuelva el elemento más pequeño de la pila. Implementa este método:

a) Accediendo a los atributos de *LEGPila*.
b) Usando únicamente los métodos del modelo.

Solución:

```
public interface PilaExt<E extends Comparable<E>> extends Pila<E> {
  E minimo();
}
```

a)

```
public class LEGPilaExt<E extends Comparable<E>> extends LEGPila<E>
       implements PilaExt<E> {

  public E minimo() {
    NodoLEG<E> aux = tope;
    E min = null;
    while (aux != null) {
      if (min == null || aux.dato.compareTo(min) < 0) min = aux.dato;
      aux = aux.siguiente;
    }
    return min;
  }

}
```

b)

```
public class LEGPilaExt<E extends Comparable<E>> extends LEGPila<E>
       implements PilaExt<E> {

  public E minimo() {
    if (esVacia()) return null;
    E dato = desapilar();
    E minResto = minimo();
    apilar(dato);
    if (minResto == null || dato.compareTo(minResto) < 0) return dato;
    return minResto;
  }

}
```

Ejercicio 5.- Amplia la funcionalidad de la EDA *Cola* vía herencia para añadir un nuevo método que invierta el orden de los elementos la cola. Implementa este método:

a) Accediendo a los atributos de *ArrayCola*.
b) Usando únicamente los métodos del modelo.

Solución:

```
public interface ColaExt<E> extends Cola<E> {
  void invertir();
}
```

a)

```
public class ArrayColaExt<E> extends ArrayCola<E> implements ColaExt<E> {

  public void invertir() {
    int i = primero, j = fin;
    for (int cont = 0; cont < talla/2; cont++) {
      E aux = elArray[i];
      elArray[i] = elArray[j];
      elarray[j] = aux;
      if (++i == elArray.length) i = 0;
      if (--j == -1) j = elArray.length - 1;
    }
  }

}
```

b)

```
public class ArrayColaExt<E> extends ArrayCola<E> implements ColaExt<E> {

  public void invertir() {
    if (!esVacia()) {
      E tmp = desencolar();
      invertir();
      encolar(tmp);
    }
  }

}
```