Grado en Ingeniería Informática y Doble Grado en Informática y Administración de Empresas	***MODELO A*** Asignatura Estructura de Datos y Algoritmos **24 de Marzo de 2014.** **SEGUNDO EXAMEN PARCIAL**

Nombre: .....

Apellidos: .....

Grupo: .....

LEA ATENTAMENTE ESTAS INSTRUCCIONES ANTES DE COMENZAR LA PRUEBA

1. Es necesario poner todos los datos del alumno en el cuadernillo de preguntas (este documento). Use un bolígrafo para rellenarlos.
2. El examen está compuesto por 5 Preguntas.
3. Solamente se evaluará la contestación en este cuadernillo de preguntas.
4. Cuando finalice la prueba, se deben entregar el enunciado del examen y cualquier hoja que haya empleado.
5. No está permitido salir del aula por ningún motivo hasta la finalización del examen.
6. Desconecten los móviles durante el examen.
7. La duración del examen es de **2 horas**.
8. **Todos los métodos necesarios de las librerías EdaLib son públicos y para simplificar la implementación de sus soluciones puede considerar que los atributos de las clases son públicos.**

NO PASE DE ESTA HOJA hasta que se le indique el comienzo del examen

Problema 1 (2 puntos).

Dado un polinomio formal $\mathbf{P(x)=a_0+a_1x+a_2x^2+a_3x^3+...+a_nx^n}$ (la siguiente figura muestra la especificación formal del tipo abstracto de datos polinomio).

```
public interface iPolinomio {
    /**
     * devuelve el grado del polinomio
     * Ejemplos:
     * P(x)=3, su grado es 0.
     * P(x)=x+1 su grado es 1
     * P(x)=x^2+x, su grado es 2.
     * P(x)=x^4+1, su grado es 4.
     */
    public int getGrado();
    /**
     * devuelve el coeficiente del término de grado n
     * Dado el polinomio, P(x)=2x^3+2
     * Por ejemplo, getCoeficiente(3)=2,
     * getCoeficiente(2)=0,
     * getCoeficiente(1)=0,
     * getCoeficiente(0)=2
     */
    public int getCoeficiente(int n);
    /**
     * modifica el coeficiente del término de grado n al nuevo valor newValue
     * Por ejemplo, dado el polinomio   P(x)=x^3+1
     * si ejecutamos setCoeficiente(3,4) => P(x)=4x^3+1;
     */
    public void setCoeficiente(int n, int newValue);
    /**
     *  calcula el valor del polinimio para un determinado valor x
     *  Por ejemplo, dado el polinomio  P(x)=x^3+1
     *  calcularValor(1)=2
     */
    public int calcularValor(int x);
    /**
     * devuelve un polinomio que es la suma del polinomio objeto y el polinomio
     * que se recibe por parámetro
     * Por ejemplo, P(x)=x^3+1, Q(x)=x^4+4x^3+7x+5
     * entonces el resultado es x^4+5x^3+7x+5
     */
    public iPolinomio suma(iPolinomio p);
}
```

Crea una clase, Polinomio, que represente el tipo abstracto de datos polinomio. **No puedes utilizar estructuras dinámicas en su implementación**:

a) (0.25 punto) Crea los atributos que consideres necesarios para representar el polinomio y escribe un método constructor que permita inicializar los coeficientes del polinomio (dichos valores se reciben como argumento del constructor).

Solución:

```
    int[] coef;

    public Polinomio(int[] coeficientes) {
           this.coef=coeficientes;
```

```
    }
```

b) (0.50 punto) Implementa el método *getGrado()*, que devuelve el grado del polinomio.

Por ejemplo, el polinomio P(x)=5 tiene grado 0, mientras que el polinomio $P(x)=x^2+1$ tiene grado 2.

Solución:

```
public int getGrado() {
	for (int i=coef.length-1;i>=0;i--) {
		if (coef[i]!=0) return i;
	}
	return 0;
}
```

c) (0.25 punto) Implementa los método *getCoeficiente(int n)* (devuelve el coeficiente del término de grado n) y el método *setCoeficiente(int n, int newValue)* (modifica el coeficiente del término de grado n).

Por ejemplo, dado el polinomio $\mathbf{P(x)=a_0+a_1x+a_2x^2+a_3x^3+...+a_nx^n}$, el método getCoeficiente(3) devuelve a_3.
Por ejemplo, dado el polinomio $\mathbf{P(x)=a_0+a_1x+a_2x^2+a_3x^3+...+a_nx^n}$, el método setCoeficiente(3,b) da lugar a $\mathbf{P(x)=a_0+a_1x+a_2x^2+bx^3+...+a_nx^n}$.

Solución:

```
public int getCoeficiente(int n) {
	if (n>=coef.length || n<0) {
		System.out.println("término fuera de rango");
		return -1;
	}
	return coef[n];
}

public void setCoeficiente(int n, int newValue) {
	if (n>=coef.length || n<0) {
		System.out.println("término fuera de rango");
		return;
	}
	coef[n]=newValue;
```

d) (1 punto) Implementa el método *calcularValor(int x),* que recibe un valor entero y calcula el valor del polinomio para dicho valor.

Por ejemplo, $P(3)= a_0+a_13+a_23^2+b3^3+...+a_n3^n$

Nota: Puedes utilizar el método **Math.power(a,b)** para calcular el valor de un número a elevado a una potencia b.

Solución:

```
public int calcularValor(int x) {
	int result=0;
	for (int i=0;i<coef.length;i++) {
		result+=coef[i]*(int)Math.pow(x, i);
	}
```

```
        return result;
}
```

Problema 2 (2 puntos).

a) (0.5 punto) Crea una clase que implemente una lista de caracteres. La lista debe ser una lista simplemente enlazada. **Nota: la solución no puede ocupar más de una línea!!!**

Solución:

```
public class ListaCaracteres extends SList<Character> {…}
```

b) (1.5 puntos) Crea un método, **doblaVocales()**, que recorra la lista doblando aquellos nodos que contengan una vocal. Dicho en otras palabras, si el nodo visitado contiene una vocal, entonces el método crea un nodo (que almacena la misma vocal) y lo inserta justo después. Por ejemplo:

Entrada: k e p a
Salida: k e **e** p a **a**

Entrada: k,p,a,a
Salida: k,p,a,**a**,a,**a**

Nota: No se puede utilizar arrays, ArrayList, LinkedList. Sólo está permitido utilizar las clases SList y SNode.

Solución:

```
public void doblaVocales() {
    SNode<Character> nodeIt=this.firstNode;
    while (nodeIt!=null) {
          char c=nodeIt.getElement();
          if (esVocal(c)) {
                SNode<Character> newNode=new SNode<Character>(c);
                newNode.nextNode=nodeIt.nextNode;
                nodeIt.nextNode=newNode;
                nodeIt=nodeIt.nextNode;
          }
          nodeIt=nodeIt.nextNode;
    }
}
```

```
    public static boolean esVocal(char c) {

          c=Character.toLowerCase(c);
          return (c=='a' || c=='e' || c=='i' || c=='o' || c=='u' );
    }
```

Problema 3 (2 puntos).

a) (1 puntos) El método **getIndexOf(E e)** de la clase **DList<E>** devuelve la posición del primer nodo que contiene al objeto *e*. En este método, la búsqueda se realiza de izquierda a derecha desde el nodo *header*. Implementa un nuevo método **getRevIndexOf** (E e) que devuelva la posición del primer nodo que contiene al objeto *e* pero realizando la búsqueda de derecha a izquierda y comenzando en el nodo *tailer*. **Para obtener la posición del nodo, tendrás que tener en cuenta que la posición del primer nodo (*header.next*) es 0 y la posición del último nodo (*tailer.prev*) es *getSize()-1*.**

Ejemplo:
Sea L-> a, b, c, d, b, e, a.
getIndexOf(b)=1
getRevIndexOf(b)=4

Solución:

```
public int getRevIndexOf(E elem) {
	int index = this.getSize()-1;
	DNode<E> nodeIt = trailer.getPreviousNode();
	while (nodeIt != header) {
		if (nodeIt.getElement().equals(elem)) {
			return index;
		}
		nodeIt = nodeIt.getPreviousNode();
		--index;
	}
	return -1;
}
```

b) Escribe el código del método **insertAt(E e, int index)** de la clase **DList<E>**.

Solución:

```
public void insertAt(int index, E elem) {
		DNode<E> newNode = new DNode<E>(elem);
		int i = 0;
		for (DNode<E> nodeIt = header; nodeIt != trailer; nodeIt =
nodeIt.nextNode) {
			if (i == index) {
				newNode.nextNode = nodeIt.nextNode;
				newNode.previousNode = nodeIt;
				nodeIt.nextNode.previousNode = newNode;
				nodeIt.nextNode = newNode;
				return;
			}
			++i;
		}
		System.out.println("DList: Insertion out of bounds");
	}
```

Problema 4 (2 puntos).

Escribir un programa en Java que reciba como argumentos dos colas ordenadas (por ejemplo, cola1=[2,3,6,8,9], cola2=[0,1,4,5,7]) , y devuelva un cola que mezcle ambas colas en otra cola ordenada.

Solución:

```
/**
 * Método que mezcla dos colas ordenadas
 */
public static SQueue<Integer> mezclarColas(SQueue<Integer> c1, SQueue<Integer> c2) {
    System.out.println("Cola c1: " + c1.toString());
    System.out.println("Cola c2: " + c2.toString());

    SQueue<Integer> c3=new SQueue<Integer>();


    while (!c1.isEmpty() && !c2.isEmpty()){
        if (c1.front()<c2.front()) c3.enqueue(c1.dequeue());
        else c3.enqueue(c2.dequeue());
    }
    //puede que en la c1 todavía queden elementos
    while (!c1.isEmpty()){
        c3.enqueue(c1.dequeue());
    }

    //puede que en la c2 todavía queden elementos
    while (!c2.isEmpty()){
        c3.enqueue(c2.dequeue());
    }

    return c3;
}
```

Problema 5 (2 puntos).

Ejercicios de Recursividad y Complejidad.

a) Rellena la siguiente tabla con las complejidades de las operaciones

	SList	DList
addLast	O(n)	O(1)
removeLast	O(n)	O(1)
insertAt	O(n)	O(n)
getIndexOf	O(n)	O(n)

b) ¿Cuál es la complejidad del método enqueue de la clase SQueue?, ¿y del método dequeue?. ¿Estos métodos tienen menor complejidad en la clase DQueue?.

Solución: La complejidad de ambos métodos es O(1) tanto en la clase SQueue como en la clase DQueue.

c) Escribe un método recursivo que reciba dos números enteros, a y b (b debe ser positivo) y que calcule la potencia de a elevado a b.

Solución:

```
public static int potencia(int a, int b) {
   if (b==0) return 1;
   else return a*potencia(a,b-1);
}
```

d) Escribe un método recursivo que reciba un número entero positivo n y devuelva su factorial.

Solución:

```
public static int factorial(int n) {
      if (n==1) return 1;
      else return n*factorial(n-1);
}
```