## *Ejercicios Tema 7. POO I. Estructuras dinámicas. Arrays, Pilas y Colas*

*Todos los ejercicios de este boletín deben estar dentro de un **package** llamado **tema07pilascolas** (siguiendo las reglas que hemos visto en clase, ejemplo: com.germangascon.tema07pilascolas).*

1. Implementar una clase llamada **DynamicArray** que permita manejar **Arrays dinámicos**. Para ello deberá definir las siguientes operaciones:
   a) boolean **add**(double value): añadirá el valor indicado al final del array. Si el array está lleno, deberemos "expandirlo" creando uno nuevo con el doble de tamaño, copiando todos los elementos al nuevo array y haciendo que nuestro array apunte al nuevo array.
   b) boolean **add**(int index, double value): añadirá el valor indicado en la posición indicada, desplazando una posición todos los elementos que queden a la derecha. Si el array está lleno, "expandiremos" igual que en el apartado anterior.
   c) double **remove**(int index): elimina del array el elemento que ocupa la posición indicada, desplazando una posición hacia la izquierda todos los elementos que queden a su derecha.
   d) double **remove**(double value): elimina del array la primera ocurrencia que se encuentre del valor recibido como parámetro, desplazando una posición hacia la izquierda todos los elementos que queden a su derecha.
   e) double **get**(int index): obtiene el elemento de la posición indicada
   f) boolean **set**(int index, double value): establece el valor del elemento que ocupa la posición index.
   g) int **size**(): devuelve el tamaño efectivo del array, es decir, cuantos elementos hay en el array.

2. Implementar la Clase **Pila** utilizando Arrays con los métodos vistos en el Anexo y posteriormente crear varios casos de prueba desde la clase Main.

3. Utilizando la clase **Pila** creada en el ejercicio anterior, crea un programa que analice el c**ódigo fuente de un programa** y determine si la cantidad y el orden de los **paréntesis**, **llaves** y **corchetes** es correcto.
   Aún no hemos visto como manipular archivos en Java, pero recuerda, podemos **redirigir la entrada estándar** y leer el fichero mediante un **Scanner**.
   El proceso a seguir será el siguiente: cada vez que nos encontremos un paréntesis/llave/corchete de *apertura* haremos una operación **push** y cada vez que nos encontremos con un paréntesis/llave/corchete de **cierre** haremos una operación **pop** y comprobaremos si el valor que leído (cierre) se corresponde con la etiqueta de apertura del valor extraído de la pila (pop).
   Para llevar a cabo esta comparación, deberás establecer un mecanismo que permita relacionar valores de apertura y de cierre. **Si alguno no coincide**, entenderemos que el código fuente del programa **está mal formado**.

4. Utilizando la clase Pila crea un programa que permita determinar si un documento **html** está bien

formado. Para saber si está bien formado utilizaremos las siguientes condiciones:

a) Todas las etiquetas deben de tener etiqueta de apertura y de cierre.
b) El orden de cierre de las etiquetas debe de ir de la más interna a la más externa.

La forma de proceder es similar al ejercicio anterior, recibiremos el archivo por la **entrada estándar** y lo leeremos mediante un **Scanner**. Cada vez que nos encontremos con una **etiqueta de apertura** haremos una operación **push** y, al leer una **etiqueta de cierre** haremos una operación **pop** y comprobaremos si efectivamente la etiqueta leída es la etiqueta de cierre de la etiqueta extraída de la pila. Algunos consejos que te pueden ayudar:

- Antes de procesar el documento, **elimina los espacios en blanco sobrantes**, sólo deja uno.
- **Ignora** en el procesado aquellas **etiquetas con autocierre**, es decir, las etiquetas que acaban con /> como por ejemplo <br />, <img src="….." />
- **Ignora** el contenido de los **atributos** ya que dentro de ellos pueden aparecer caracteres como "/" que pueden confundirse el cierre de etiquetas.
- Al procesar las etiquetas **quédate sólo con su nombre** ignorando los caracteres '<' y '>' así cuando obtengas la de cierre te será fácil hacer la correspondencia. Por ejemplo, si te encuentras con una etiqueta  sólo deberás guardar "p" en la pila, y cuando leas  te quedarás con su valor "p" y lo compararás con el extraído "p".

Si nos **encontramos con alguna etiqueta distinta** de la que obtenemos con el pop entonces el **documento está mal formado**. Si llegamos al final del documento y **no hemos encontrado diferencias**, diremos que el **documento está bien formado**.

5. Utilizando la clase **Pila** crea un programa que permita analizar expresiones algebraicas en Notación Polaca Inversa (RPN). La RPN es un método algebraico alternativo de introducción de datos donde primero están los operandos y después viene el operador que va a realizar los cálculos sobre ellos.

Por ejemplo: la expresión "20 + 4" se escribiría como "20 4 +".
Otro ejemplo: la expresión "12 – (3 + 5) * 2" se escribiría como "12 3 5 + 2 * -".

La expresión completa la guardaremos en un Array, donde cada posición del Array será un operando o un operador.
Para evaluar la expresión necesitaremos la estructura Pila donde iremos guardando resultados intermedios. Para ello recorreremos el Array de izquierda a derecha, y para cada elemento, realizaremos los siguiente:

a) Si se trata de un **operando** simplemente **lo apilamos** (push) en la Pila.
b) Si se trata de un **operador**, **desapilamos** (pop) de la Pila **dos operandos**, realizamos la operación indicada por el operador sobre los dos operandos extraídos y el resultado lo apilamos.

Cuando hayamos recorrido todo el Array el último valor que queda en la Pila es el resultado de la expresión.

6. Implementar la Clase **Cola** mediante estructuras estáticas (Arrays) con los métodos vistos en el Anexo y posteriormente crear varios casos de prueba desde la clase Main.

7. Utilizando la clase **Cola** creada en el ejercicio anterior y reutilizando código del ejercicio 7 del anterior boletín de ejercicios, crea un programa para simular la cola de espera de una consulta médica. El funcionamiento debe ser el siguiente:
   a) Un paciente llega al centro de salud introduce su SIP y el sistema le muestra un menú con los médicos disponibles. El paciente selecciona el médico que desee y el sistema lo pone en cola de dicho médico y le asigna un código alfanumérico de longitud 3 para preservar su privacidad. Cuando el paciente es "llamado" se hace mediante este código.
   b) Una vez el paciente ha obtenido código, espera hasta que dicho número aparezca en la pantalla.

   Para realizar la simulación de llamada de pacientes añadiremos una opción en el menú que al pulsarla solicitará el médico y llamará (mostrará en pantalla el código del paciente y la puerta donde debe dirigirse) al siguiente paciente de la Cola de espera de dicho médico.

   El menú de la aplicación quedará de la siguiente forma:

   **********
   ** H O S P I T A L **
   **********
   1. Introducir SIP
   2. Llamar a paciente
   3. Consultas ...

   Al pulsar la opción de consultas mostraremos el siguiente submenú:

   **********
   ** C O N S U L T A S **
   **********
   1. Citas de un paciente
   2. Citas de un médico
   3. Mostrar todos los pacientes
   4. Mostrar todas las citas

   Para facilitar la depuración y pruebas del programa, se debe generar un conjunto de casos de prueba (médicos, pacientes y citas).

8. Crea una nueva clase llamada **GenericDynamicArray** que implemente un **array dinámico** utilizando **Generics** tal y como hemos visto en clase.