# Práctica 2

## Datos generales:

| | |
|---|---|
| **Nombre de la Práctica** | Arreglos |
| **Nombre de la carrera** | Ingeniería de Software |
| **Nombre de la materia** | Estructuras de Datos |
| **Número y nombre de Unidad(es) temática(s)** | 1. Introducción a las estructuras de datos y estructuras fundamentales. |
| **Docente que imparte la materia** | Aldonso Becerra Sánchez |
| | |
| **Fecha de entrega para los alumnos** | 18-agosto-2021 |
| **Fecha de entrega con extensión y penalización** | 19-agosto-2021 |
| **Fecha de elaboración** | 18-agosto-2021 |

| | |
|---|---|
| **Objetivo de la tarea** | Creación de TDA arreglo (ArregloDatos) para su posterior uso en aplicaciones comunes. |
| **Tiempo aproximado de realización** | 3 horas |
| **Introducción** | Existe diversidad de usos que se les puede dar a los arreglos. Desde este punto de vista, los arreglos propician que muchos planteamientos puedan tener una solución sencilla si se llevan a cabo con la ayuda de ellos. |

## Referencias que debe consultar el alumno (si se requieren):

### Referencia 1:

**1.**Cairo, Osvaldo; Guardati, Silvia. Estructura de Datos, Tercera Edición. McGraw-Hill, México, Tercera Edición, 2006.

**Referencia 2:**

**2.**Mark Allen Weiss. Estructura de datos en Java. Ed. Addison Wesley.

**Referencia 3:**

**3.** Joyanes Aguilar, Luis. Fundamentos de Programación. Algoritmos y Estructuras de Datos. Tercera Edición, 2003. McGraw – Hill.

**Actividades que debe realizar el alumno:**

**Actividad inicial:**

Lea primero toda la práctica. No inicie a programar sin leer todo cuidadosamente primero. Recuerde que debe generar el reporte en formato IDC.

**Actividad 1:**

Primero genere la **Introducción**.

**Actividad 2:**

Complete el TDA llamado "ArregloDatos", el cual debe tener atributos y métodos necesarios para manipularlo en inserción, búsqueda, modificación y eliminación de forma desordenada.

Además agregue los métodos de:

- `[declarado en interface ListaDatos] public boolean esIgual(Object listaDatos2):` indica si la lista actual es igual a la listaDatos2. La igualdad se determina por el contenido de cada elemento en la posición correspondiente al mismo índice. Debe validar que listaDatos2 sea un ArregloDatos.
- `[declarado en interface VectorDatos] public Object obtener(int indice):` regresa el Objeto de la posición "indice". Recuerde validar que el índice que se pasa como argumento debe estar en un rango válido, en caso contrario regrese null.
- `[declarado en interface ListaDatos] public boolean cambiar(Object elementoViejo, Object elementoNuevo, int numVeces):` modifica el elemento viejo por el elemento nuevo haciendo una búsqueda y modificación del número de veces de ocurrencias del elemento encontrado indicado por numVeces. Regrese verdadero si hizo alguna modificación. No olvide las validaciones de rangos de índices.

- [declarado en interface VectorDatos] `public boolean cambiar(int indice, Object elemento):` modifica el elemento de la posición indicada por "indice". "elemento" es el nuevo valor a colocar. Realice el proceso de validación para que no permita cambiar nodos inexistentes en el "índice". Regrese verdadero si sí pudo o falso si no se pudo.
- [declarado en interface VectorDatos] `public boolean cambiarArregloDatos(ArregloDatos indicesBusqueda, ArregloDatos elementosNuevos):` modifica el elemento del arreglo actual en las posiciones de "indicesBusqueda" por el contenido de "elementosNuevos". De esta forma se tendrán nuevos valores en cada una de las posiciones del arreglo viejo por cada una de las posiciones del arreglo nuevo tomando como base los "indicesBusqueda". Regrese verdadero si sí pudo o falso si no se pudo o no debido a las dimensiones o índices inválidos, por ejemplo.
- [declarado en interface ListaDatos] `public ArregloDatos buscarValores(Object elemento):` busca dentro de un arreglo los elementos indicados por elemento, si existen muchos elementos o en su caso uno; debe guardar en un arreglo (que devolverá) las posiciones de todas y cada una de las ocurrencias del elemento buscado.
- [declarado en interface VectorDatos] `public Object eliminar(int indice):` elimina un elemento del arreglo en una posición específica, regresando el elemento eliminado.
- [declarado en interface ListaDatos] `public Object eliminar():` regresa el objeto de la última posición del ArregloDatos.
- [declarado en interface Lista] `public void vaciar():` vacía el contenido de un arreglo.
- [declarado en interface ListaDatos] `public boolean agregarLista(Object listaDatos2):` agrega al final de la lista actual (en este caso el TDA ArregloDatos) el contenido de la listaDatos2 (en este caso otro arreglo). Debe validar que listaDatos2 sea un ArregloDatos.
- [declarado en interface ListaDatos] `public void invertir():` invierte el orden de los elementos de un arreglo.
- [declarado en interface ListaDatos] `public int contar(Object elemento):` contar cuántos elementos hay en el arreglo con el contenido igual a valor especificado por elemento.
- [declarado en interface ListaDatos] `public boolean eliminarLista(Object listaDatos2):` elimina cada elemento de listaDatos2 que se encuentre en la lista actual (en este caso el TDA ArregloDatos). Debe validar que listaDatos2 sea un ArregloDatos.
- [declarado en interface ListaDatos] `public void rellenar(Object elemento, int cantidad):` rellena todos los elementos indicados por "cantidad" del arreglo con el "elemento" indicado.

Validar que cantidad no se pase del tamaño de la estructura, en caso que sea estática; sino sólo rellenar hasta la capacidad máxima.

- `[declarado en interface ListaDatos] public Object clonar():` regresa una copia de la lista actual.
- `[declarado en interface ListaDatos] public Object subLista(int indiceInicial, int indiceFinal):` regresa una lista con los elementos indicados con las posiciones inicial y final. Valide los rangos.
- `[declarado en interface VectorDatos] public Object redimensionar(int maximo):` redimensiona el tamaño del arreglo al nuevo tamaño indicado por máximo. Si el tamaño es menor, los elementos sobrantes deben ser eliminados. Si el tamaño es mayor, los datos anteriores deben conservarse.

No olvide colocar los comentarios de documentación para javadoc. Ya que el reporte IDC debe tener el anexo de la documentación.

Esta actividad debe entrar en la parte de **Desarrollo**.

**Actividad 3:**

Pruebe el funcionamiento del programa de la actividad 2 con todo y sus capturas de pantalla.

**Actividad 4:**

Realice la sección de **Código agregado** (diagrama de clases UML).

**Actividad 5:**

Realice la sección de **Pre-evaluación** (use los lineamientos establecidos).

**Actividad 6:**

Finalmente haga las **Conclusiones**.

**Actividad 5:**

Enviar en http://ingsoftware.reduaz.mx/moodle

**Archivo anexo que se requiere para esta tarea (opcional):**

Dudas o comentarios: a7donso@gmail.com