|  |  |  |
| --- | --- | --- |
|  | **UNIVERSIDAD ABIERTA INTERAMERICANA** | |
|  | **FACULTAD DE TECNOLOGÍA INFORMÁTICA** | |
| **PARCIAL NRO 1** | |
| **Asignatura: Programación I** | |
| **Mesa: O**. Examen O. Recuperatorio de Materia O. Examen Final | |
| **Localización:** Uai-Online |  |
|  | **Turno:** O. Mañana **O**. Noche | **Fecha:** 8/10/20 |

|  |  |
| --- | --- |
| **Alumno:** Julián Emilio Ryan | **Legajo:** B00044969-T4 |
| **Obs:** | **Nota:** |

Requisitos para aprobar: Para que el parcial esté aprobado el alumno deberá tener correctamente desarrolladas el 60% de la teoría y resuelto el ejercicio práctico.

Recomendaciones:

1. Lea todo el parcial antes de comenzar a responder.
2. Desarrolle una redacción clara y precisa contestando lo que la pregunta requiere.
3. Observe la ortografía ya que la misma es parte del parcial.
4. Si considera que no comprende alguna consigna antes de comenzar consulte a su profesor.

# Teoría

* 1. ¿Qué es una clase? Ejemplifique
  2. ¿Qué es un objeto? Muestre una porción de código de ejemplo relacionado con el punto anterior.
  3. ¿Cuál es la diferencia entre un atributo y una propiedad?
  4. ¿Qué es un constructor y para que se utiliza la sobrecarga de constructores?
  5. ¿Qué es una lista enlazada y qué la diferencia de un vector?
  6. ¿En qué se diferencian las estructuras de tipo Pila y Cola?
  7. ¿Por qué es conveniente separar el proyecto de librería, y utilizar solamente las clases públicas como en los ejemplos vistos en clase para las listas, colas y pilas?

# Práctica

Realizar el siguiente ejercicio en código C#:

1. Implementar por consola el uso de una clase “Cola” (Con tamaño máximo establecido por el usuario) que permita ejecutar las siguientes operaciones: Enqueue, Dequeue y Peek, Vacia y LLena. El objeto que podrá “encolarse” será del Tipo Alumno y contendrá los siguientes datos: Nombre: String; Apellido: String; FechaNac: DateTime.
2. Implementar un mecanismo de búsqueda en la propia clase “Cola” que devuelva el primer elemento encontrado a partir del parámetro Apellido. Este mecanismo solo devuelve un objeto de tipo Alumno, no hay que modificar ni eliminar ningún elemento de la Queue.

Para ambos puntos utilizar nodos simplemente enlazados. Realizar casos de prueba desde la consola (Método main()) para validar que funcionen los métodos solicitados y validando los límites establecidos tanto para funcionalidad vacía o llena.

Desarrollo:

Teoría

1. ¿Qué es una clase? Ejemplifique

Las Clases son la representación simbólica de los Objetos dentro de la Programación Orientada a Objetos y que describen campos, propiedades, métodos y eventos con los que se va a trabajar. Por ejemplo, una clase podría ocupar el plano de un motor y los objetos a cada motor que se producen en una fábrica serían los objetos.

Ejemplo:

public class Paciente

{

public String Nombre { get; set; }

public int Temperatura { get; set; }

public Paciente(int temperatura = 36)

{

this.Temperatura = temperatura;

}

}

1. ¿Qué es un objeto? Muestre una porción de código de ejemplo relacionado con el punto anterior.

Un Objeto dentro de la Programación Orientada a Objetos se refiere a una entidad de la vida real, es decir, alguno de los objetos únicos que pertenecen al problema y con el se puede interactuar. Cada objeto consta de un estado (es decir, unos atributos con unos valores concretos) y de un comportamiento (es decir, tiene funcionalidades o sabe hacer unas acciones concretas), que a su vez constan respectivamente de datos almacenados y de tareas realizables durante el tiempo de ejecución. Dentro de este paradigma, los objetos son creados instanciando una clase.

Ejemplo:

Lista listaPacientes = new Lista();

Paciente paciente1 = new Paciente(37);

paciente1.Nombre = "Lucía";

listaPacientes.AgregarAlComienzo(paciente1);

Paciente paciente2 = new Paciente();

paciente2.Nombre = "Martín";

listaPacientes.AgregarAlComienzo(paciente2);

Paciente paciente3 = new Paciente();

paciente3.Nombre = "Jorge";

listaPacientes.AgregarAlComienzo(paciente3);

listaPacientes.Recorrer();

1. ¿Cuál es la diferencia entre un atributo y una propiedad?

Un atributo es una característica de un tipo de objeto, definida en una clase y representada como una variable cuyo valor diferencia un objeto de otro y determina su apariencia, estado u otra cualidad. Una propiedad es un método que permite modificar esa variable mediante una entrada u obtenerla mediante una salida. La diferencia principal es que una propiedad es una forma de acceder al contenido de un atributo, tanto para consultar su valor como para modificarlo.

1. ¿Qué es un constructor y para que se utiliza la sobrecarga de constructores?

Un constructor es un método especial que podemos utilizar con las clases. Éste generalmente es usado para inicial los valores de los datos con los que trabajará el objeto. La forma como lo utilizamos con las clases es equivalente a la forma como lo utilizamos con las estructuras en el capítulo anterior. El constructor es un método especial y tiene ciertas características que lo distinguen de los demás métodos. Su primera característica es que tiene el mismo nombre de la clase y su segunda característica más importante es que no tiene tipo, es decir, que no solamente no regresa nada, sino que no tiene tipo alguno

El constructor puede ser sobrecargado, es decir, podemos tener más de una versión del constructor para poder seleccionar cómo se inicializarán los datos del objeto dependiendo del tipo de constructor que utilicemos.

1. ¿Qué es una lista enlazada y qué la diferencia de un vector?

Una lista enlazada es una de las estructuras de datos fundamentales, y puede ser usada para implementar otras estructuras de datos. Consiste en una secuencia de nodos, en los que se guardan campos de datos arbitrarios y una o dos referencias, enlaces o punteros al nodo anterior o posterior.

Una lista enlazada es un tipo de dato autorreferenciado porque contienen un puntero o enlace (en inglés link, del mismo significado) a otro dato del mismo tipo. Las listas enlazadas permiten inserciones y eliminación de nodos en cualquier punto de la lista en tiempo constante (suponiendo que dicho punto está previamente identificado o localizado), pero no permiten un acceso aleatorio. Existen diferentes tipos de listas enlazadas: listas enlazadas simples, listas doblemente enlazadas, listas enlazadas circulares y listas enlazadas doblemente circulares.

El principal beneficio de las listas enlazadas respecto a los vectores convencionales es que el orden de los elementos enlazados puede ser diferente al orden de almacenamiento en la memoria o el disco, permitiendo que el orden de recorrido de la lista sea diferente al de almacenamiento, sin embargo, esto hace que las listas consuman una cantidad de memoria RAM mayor.

1. ¿En qué se diferencian las estructuras de tipo Pila y Cola?

Las estructuras de tipo Pila y Cola se diferencian, por un lado, en su modelo de entrada y salida, es decir, una Pila tiene un modelo LIFO (Last-In / First-Out), el cual quiere decir que el último elemento en entrar a la Pila será el primero en salir. En cambio, una cola tiene un modelo FIFO (First-In / First-Out), el cual quiere decir que el primer elemento en entrar, será el primer elemento en salir. Y, por otro lado, en una pila los elementos se agregan y se eliminan en el mismo extremo. En cambio, en una cola los elementos se agregan de un extremo de la cola llamado "final", y se eliminan del otro extremo de la cola llamado "frente".

1. ¿Por qué es conveniente separar el proyecto de librería, y utilizar solamente las clases públicas como en los ejemplos vistos en clase para las listas, colas y pilas?

Es conveniente separar el proyecto de librería, y utilizar solamente las clases públicas como en los ejemplos vistos en clase para las listas, colas y pilas ya que, de esta manera la clase de nodos (en los ejemplos vistos en clase, aunque podría ser otra u otras clases), quedan internas, sin acceso al usuario, protegiendo la integridad del código, aumentando la seguridad y disminuyendo la posibilidad de que éste se corrompa.

Práctica

Realizar el siguiente ejercicio en código C#:

1. Implementar por consola el uso de una clase “Cola” (Con tamaño máximo establecido por el usuario) que permita ejecutar las siguientes operaciones: Enqueue, Dequeue y Peek, Vacia y LLena. El objeto que podrá “encolarse” será del Tipo Alumno y contendrá los siguientes datos: Nombre: String; Apellido: String; FechaNac: DateTime.
2. Implementar un mecanismo de búsqueda en la propia clase “Cola” que devuelva el primer elemento encontrado a partir del parámetro Apellido. Este mecanismo solo devuelve un objeto de tipo Alumno, no hay que modificar ni eliminar ningún elemento de la Queue.

Para ambos puntos utilizar nodos simplemente enlazados. Realizar casos de prueba desde la consola (Método main()) para validar que funcionen los métodos solicitados y validando los límites establecidos tanto para funcionalidad vacía o llena.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Parcial1

{

class Program

{

static void Main(string[] args)

{

//declaración de variables iniciales

int tamañoMaximo = 0;

int opc = 0;

//comienzo del programa y solicitud de tamaño máximo de la cola

Console.WriteLine("\*Software de Cargado de Colas de Alumnos\*\n");

Console.WriteLine("Ingrese la cantidad de alumnos con la que se va trabajar: \n");

tamañoMaximo = Int32.Parse(Console.ReadLine());

//creación de la cola

Cola ColaAlumnos = new Cola(tamañoMaximo);

ColaAlumnos.TamMaximo(tamañoMaximo);

//ejecución del menu para operar la cola mientras la opción del usuario sea distinta de 0

do

{

try //utilización de try y catch para evitar cierres inoporturnos y errores del programa

{

Console.WriteLine("\n Ingrese la accion que desea realizar: ");

Console.WriteLine("1) Cargar alumnos en la Cola de Alumnos");

Console.WriteLine("2) Mostrar al primero alumno de la Cola de Alumnos");

Console.WriteLine("3) Buscar alumno por Apellido");

Console.WriteLine("4) Corroborar si la Cola de Alumnos esta vacia");

Console.WriteLine("5) Corroborar si la Cola de Alumnos esta llena");

Console.WriteLine("6) Borrar el primer alumno agregado a la Cola de Alumnos");

Console.WriteLine("7) Mostrar la Cola de Alumnos"); //para corroboración

Console.WriteLine("Ingrese la accion que desea realizar (cero para salir): ");

opc = Int32.Parse(Console.ReadLine()); //lectura de la opción

}

catch (Exception error)

{

Console.WriteLine(error.Message);

}

switch (opc) //switch para las diferentes opciones del menu

{

case 1: //creación un nuevo alumno, solicitud de los datos y encolado del mismo por función Queue

Alumno alumno = new Alumno();

if (!ColaAlumnos.IsFull()) //corroboración de que la cola tiene espacio y solicitud de datos

{

try

{

Console.WriteLine("\nIngrese el nombre del alumno:");

alumno.Nombre = Console.ReadLine();

}

catch (Exception error)

{

Console.WriteLine(error.Message);

}

try

{

Console.WriteLine("Ingrese el apellido del alumno:");

alumno.Apellido = Console.ReadLine();

}

catch (Exception error)

{

Console.WriteLine(error.Message);

}

try

{

Console.WriteLine("Ingrese la fecha de nacimiento del alumno (dd/mm/aaaa):");

alumno.FechaNac = Convert.ToDateTime(Console.ReadLine());

}

catch (Exception error)

{

Console.WriteLine(error.Message);

}

ColaAlumnos.Enqueue(alumno); //funcion de encolar al alumno

}

else //mensaje en caso de que no haya datos

{

Console.WriteLine("La cola se encuentra completa!\n");

}

break;

case 2:

ColaAlumnos.Peek(); //función directa que muestra el alumno del tope

break;

case 3:

if (ColaAlumnos.IsEmpty() != true) //funcion de busqueda, corrobora primero que la cola tenga datos

{

//definición de variables y solicitud del apellido como criterio para la busqueda

string apellidoBusq;

Alumno Encontrado = new Alumno();

Console.WriteLine("Ingrese el apellido del alumno que desea buscar: ");

apellidoBusq = Console.ReadLine();

Encontrado = ColaAlumnos.Search(apellidoBusq); //implementación de la función de busqueda con devolución del primer objeto alumno que coincide con el criterio

if (Encontrado != null) //en caso de que se haya encontrado un apellido igual al buscado se imprimen los datos del alumno

{

Console.WriteLine("Los datos del alumno son: {0} {1} {2}", Encontrado.Nombre, Encontrado.Apellido, Encontrado.FechaNac.ToString("dd/mm/yyyy"));

}

else //mensaje en caso de que no se encuentren apellidos iguales

{

Console.WriteLine("No se encontraron alumnos con ese apellido \n ");

}

}

else //mensaje en caso de que no se encuentren alumnos

{

Console.WriteLine("La Cola de Alumnos se encuentra vacia \n");

}

break;

case 4:

//ColaAlumnos.IsEmpty();

if (ColaAlumnos.IsEmpty() != true) //corroboración del estado de la cola e impresión en caso de que NO este vacia

{

Console.WriteLine("La Cola no esta vacia. Hay lugares ocupados\n");

}

else //En caso de que SI este vacia

{

Console.WriteLine("La Cola de Alumnos se encuentra vacia \n");

}

break;

case 5:

//ColaAlumnos.IsFull();

if (ColaAlumnos.IsFull() != true) //corroboración del estado de la cola e impresión en caso de que NO este llena

{

Console.WriteLine("La Cola de Alumnos no esta llena. Hay lugares disponibles en \n");

}

else //En caso de que SI este llena

{

Console.WriteLine("La Cola de Alumnos esta llena \n");

}

break;

case 6:

ColaAlumnos.Dequeue(); //función directa de eliminar primer alumno de la cola

break;

case 7:

ColaAlumnos.Show(); //función de mostrar alumnos

break;

default:

Console.WriteLine("Ingreso un valor incorrecto, vuelva a intentar");

break;

}

} while (opc != 0);

Console.WriteLine("A seleccionado salir del Software de Cargado de Colas de Alumnos. Hasta luego!\n\n");

Console.ReadLine();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Parcial1

{

internal class Nodo //clase de creación de nodos con sus propiedades

{

public Alumno Alumno { get; set; }

public Nodo Siguiente { get; set; }

public Nodo(Alumno alumno)

{

this.Alumno = alumno;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Parcial1

{

public class Alumno //clase de creación de alumno con sus propiedades

{

public string Nombre { get; set; }

public string Apellido { get; set; }

public DateTime FechaNac { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Parcial1

{

class Cola

{

//iniciación de variables

private Nodo Primero;

private Nodo Ultimo;

public int tamañoMaximo = 0;

private int tamañoActual = 0;

//tamaño máximo ingresado por el usuario, creo

public int TamMaximo(int tamañoMaximo)

{

return tamañoMaximo;

}

//constructor de la cola

public Cola(int tamañoMaximo = 0)

{

Primero = null;

Ultimo = null;

this.tamañoMaximo = tamañoMaximo;

}

//metodo para determinar si la cola esta vacia

public bool IsEmpty()

{

return tamañoActual == 0;

}

//metodo para determinar si la cola esta llena

public bool IsFull()

{

return tamañoMaximo > 0 && tamañoMaximo == tamañoActual;

}

//metodo para poner alumnos en la cola

public void Enqueue(Alumno alumno)

{

Nodo Nuevo = new Nodo(alumno); //creación de un nuevo nodo

if (!IsFull()) //corroboración del estado de la cola

{

//Nodo nodo = new Nodo(alumno);

if (Primero != null) //en caso de que el alumno no sea el único, se colocan detras en la cola los siguientes

{

Ultimo.Siguiente = Nuevo;

Nuevo.Siguiente = null;

Ultimo = Nuevo;

tamañoActual++;

}

else //en caso de que el alumno sea el único

{

Primero = Nuevo;

Primero.Siguiente = Nuevo;

Ultimo = Nuevo;

tamañoActual++;

}

}

else

{

Console.WriteLine("La cola esta llena!\n");

}

}

public void Show() //metodo para mostrar los elementos de la cola

{

if (!IsEmpty())

{

Nodo Mostrar = Primero;

while (Mostrar != null) //ciclo while hasta que el nodo llega al valor null, mientras se va imprimiendo la info de los alumnos

{

Alumno AlumnoMostrado = Mostrar.Alumno;

Console.WriteLine("Los datos del alumno son: {0} {1} {2}", AlumnoMostrado.Nombre, AlumnoMostrado.Apellido, AlumnoMostrado.FechaNac.ToString("dd/mm/yyyy"));

Mostrar = Mostrar.Siguiente;

}

}

else

{

Console.WriteLine("\n La Cola de Alumnos se encuentra vacia \n");

}

}

public Alumno Peek() //metodo para mostrar el primer valor de la cola

{

if (!IsEmpty())

{

Alumno AlumnoTope = Primero.Alumno; //se utilizan los datos de alumno guardados en el primer nodo y se imprime su información

Console.WriteLine("Los datos del alumno son: {0} {1} {2}", AlumnoTope.Nombre, AlumnoTope.Apellido, AlumnoTope.FechaNac.ToString("dd/mm/yyyy"));

return AlumnoTope;

}

else

{

Console.WriteLine("\n La Cola de Alumnos se encuentra vacia \n");

return null;

}

}

public void Dequeue() //metodo para eliminar el primer nodo de la cola

{

if (!IsEmpty())

{

Alumno AlumnoEliminado = Primero.Alumno; //se utilizan los datos de alumno guardados en el primer nodo y se "pisa" o borra su información

Console.WriteLine("Los datos del alumno eliminado son: {0} {1} {2}", AlumnoEliminado.Nombre, AlumnoEliminado.Apellido, AlumnoEliminado.FechaNac.ToString("dd/mm/yyyy"));

Primero = Primero.Siguiente;

tamañoActual--;

}

else

{

Console.WriteLine("\n La Cola de Alumnos se encuentra vacia \n");

}

}

public Alumno Search(string apellidoBuscado) //método de busqueda según criterio de apellidos

{

if (!IsEmpty())

{

Nodo Buscado = Primero;

Alumno AlumnoBuscado = Buscado.Alumno;

//luego de corroborar que la cola tiene nodos se crea uno nuevo, cuyo valor de referencia es el del primer alumno en la cola

//y va a ir comprobando en los siguientes nodos si el apellido es igual o no a ingresado por el usuario

while (Buscado != null)

{

if (Buscado.Alumno.Apellido == apellidoBuscado)

{

AlumnoBuscado = Buscado.Alumno;

return AlumnoBuscado; //si es igual, devuelve a "AlumnoBuscado" como un objeto a la función Main donde se imprimen sus datos

}

else

{

Buscado = Buscado.Siguiente;//en caso de que no lo sea se sigue buscando

}

}

}

return null;

}

}

}