

PL1 – parte 1- Estructura de datos

[Subtítulo del documento]



23 de octubre de 2023

Pablo manglano redondo 03492382Q y alberto durán vadillo 09100943G

**1. Implementar las Clases:**

* **Pedido:** Representa un pedido con un ID único, número de seguimiento, DNI del cliente y urgencia. Luego también añadiremos el número de prioridad.
* **NodoPila:** Nodo para la estructura de datos de Pila.
* **NodoCola:** Nodo para la estructura de datos de Cola.
* **NodoLista:** Nodo para la estructura de datos de Lista.
* **Pila:** Implementa la Pila LIFO para almacenar pedidos. Para ello empleamos los métodos: “insertar(const Pedido& pedido);” “extraer();” “cima();” “mostrar();” “getLongitud();” “estaVacia();” “vaciarPila();”. El método insertar inserta una estructura de Pedido en una pila y aumenta 1 la longitud de la pila. El método extraer desapila la cima de la pila si no está vacía y disminuye 1 la longitud de la pila. El método cima te devuelve la cima de una pila si esta no está vacía. El método mostrar muestra por pantalla todos los Pedidos que contenga una pila (cuando no esté vacía). El método getLongitud te devuelve la longitud de una pila (0 si está vacía). El método estaVacía devuelve True si una pila está vacía y el método vaciarPila elimina los pedidos uno a uno y disminuye en de 1 en 1 la longitud de la pila hasta dejarla completamente vacía.
* **Cola:** Implementa la Cola FIFO para las estaciones de empaquetado.
* **Lista:** Implementa la Lista para almacenar pedidos ordenados por prioridad.

**2. Implementar la Clase Gestor:**

* **Gestor:** Contiene métodos para gestionar la simulación del control de pedidos, como generar, almacenar, procesar y mostrar pedidos según las opciones del menú:
* Al introducir la opción A hacemos una llamada al método “gestor.generar12Pedidos();” de la clase Gestor. Este método emplea funciones de generación de números aleatorios para crear 12 pedidos y almacénalos en la Pila. Cada pedido es una estructura en la que guardamos las variables idPedido, dniCliente, numeroSeguimiento y urgente. Se generarán de 12 en 12 y serán almacenados hasta un máximo de 48 pedidos en la pilaPedidos.
* Al introducir la opción B hacemos una llamada al método “gestor.muestraPedidos();” de la clase Gestor. Este método llama al método de la clase Pila “mostrar();” el cual se encarga de ir imprimiendo por pantalla los datos de cada pedido hasta que haya mostrado todos mientras la pilaPedidos no esté vacía.
* Al introducir la opción C hacemos una llamada al método de la clase Gestor encargado de vaciar la pilaPedidos: “gestor.borrarPedidosPila();”. Este método hace una llamada al método “vaciarPila();” de la clase Pila. Este método va eliminando pedido a pedido hasta que no queda ninguno dejando la pilaPedidos vacía.
* Al introducir la opción D hacemos una llamada al método “gestor.encolarPedidos();” de la clase Gestor. Este método nos vacía la pilaPedidos y los almacena en 4 colas: estaciónA, estaciónB, estaciónC y estaciónD. Si el pedido no es urgente, comparará las longitudes de las colas A y B con el método “getLongitud()” e introducirá el pedido no urgente en la cola que menos pedidos tenga. En caso de tener la misma longitud lo almacenará en la estaciónA. Si el pedido es urgente sucede lo mismo, pero con las estaciones C y D.
* La opción E hace una llamada al método “gestor.muestraPedidosSalasAyB();” de la clase Gestor el cual hace dos llamadas (una para estacionA y otra para estacionB) al método de la clase Cola “mostrar();”. Este método muestra mientras no esté vacía todos los pedidos uno por uno de cada una de las colas para los pedidos no urgentes (A y B).
* La opción F hace una llamada al método “gestor.muestraPedidosSalasCyD();” de la clase Gestor el cual hace dos llamadas (una para estacionC y otra para estacionD) al método de la clase Cola “mostrar();”. Este método muestra mientras no esté vacía todos los pedidos uno por uno de cada una de las colas para los pedidos urgentes (C y D).
* La opción G hace una llamada al método “gestor.borrarPedidosColas();” de la clase Gestor que hace 4 llamadas (1 para cada cola) al método “vaciarCola();”. Este método se encarga de vaciar elemento a elemento cada una de las 4 estaciones hasta que estén vacías.
* La opción H hace una llamada al método “gestor.enlistarPedidos();” el cual se encarga de almacenar todos los pedidos de las cuatro colas en dos listas, una para los pedidos urgentes y una para los pedidos no urgentes. Este es uno de los métodos más destacados del programa. Empieza declarando dos variables de tipo int: numeroSeguimientoEst y numeroSeguimientoUrg. Está compuesto de 4 bucles while (uno para cada una de las colas) los cuales mientras las colas no estén vacías llamaran al método de la clase Cola “extraer();” encargado de extraer el pedido superior de una cola. Si este pedido es urgente le asignará un número de prioridad entre 51 y 99 mientras que si no es urgente entre 1 y 49. Estos nuevos pedidos con los nuevos números de prioridad son almacenados en las dos listas (listaEstandar y listaUrgente). Al final hace una llamada al método “ordenarPorID();” de la clase lista encargada de ordenar la lista.
* La opción I hace una llamada al método de la clase Gestor encargado de mostrar todos los pedidos de la listaEstandar: “gestor.muestraPedidosEstandar();”. Este hará una llamada al método “mostrar();” de la clase Lista que mientras no esté vacía mostrará por pantalla pedido a pedido con todos sus datos.
* La opción J hace una llamada al método de la clase Gestor encargado de mostrar todos los pedidos de la listaUrgente: “gestor.muestraPedidosUrgentes();”. Este hará una llamada al método “mostrar();” de la clase Lista que mientras no esté vacía mostrará por pantalla pedido a pedido con todos sus datos.
* La opción K hace una llamada al método “gestor.buscarPedidos();” de la clase Gestor. Este es otro de los métodos más destacados del programa. El método comienza comprobando si ambas listas están vacías y en caso de estar vacías imprime un mensaje indicando que están vacías. Inicializamos dos punteros (tempEstandar y tempUrgente) apuntando al primer nodo de cada lista, dos objetos (pedidoEstandarMaxPrioridad y pedidoUrgenteMinPrioridad) y dos variables booleanas (encontradoEstandar y encontradoUrgente). Iniciamos un bucle while encargado de encontrar el pedido estándar de mayor prioridad que lo busca por toda la lista y nos lo devuelve. Iniciamos otro bucle igual pero esta vez encargado de encontrar el pedido urgente de menor prioridad. Una vez encontrados mostrará por pantalla sus correspondientes ids.
* La opción L hace una llamada al método “gestor.reiniciar();” de la clase Gestor el cual es el encargado de reiniciar el programa a su estado inicial. Para ello elimina la pilaPedidos, las 4 colas (estacionA, estacionB, estacionC y estacionD), y las 2 listas (listaEstandar y listaUrgente). Por último, crea una nueva instancia de cada una de ellas.

**3. Implementar la Función Principal (main):**

* **main:** Muestra un menú interactivo para que el usuario seleccione las opciones A-L según lo requerido. En el menú nos muestra en todo momento el número de pedidos que tiene almacenados la pilaPedidos, las 4 colas (estaciónA, estaciónB, estaciónC y estaciónD) y las 2 listas (listaEstandar y listaUrgente).

**7. Pruebas y Debugging:**

* Después de implementar cada parte, realiza pruebas exhaustivas para asegurarte de que cada funcionalidad esté trabajando según lo esperado. Utiliza la depuración para corregir cualquier error que encuentres. Al finalizar cada opción del menú íbamos comprobando que el programa no daba errores y funcionaba correctamente teniendo en cuenta sus relaciones con otras opciones como por ejemplo que no se puede mostrar los elementos de cualquiera de las estructuras si antes no se habían almacenado en ella.

**8. Documentación y Comentarios:**

* Asegúrate de documentar tu código correctamente, incluyendo comentarios descriptivos para cada clase y método que expliquen su propósito y funcionamiento.

Dificultades encontradas:  
  
- Error “does not name a type”.