Universidad Nacional de Costa Rica

Sede Interuniversitaria de Alajuela

Escuela de Informática

Curso: Estructuras de datos

Manual técnico del programa

Estudiantes:

Andrés Vega Hidalgo

Hillary Cruz Valenzuela

Profesor:

Cristopher Montero Jiménez

II Ciclo, 2021

Contenido

Sucursal

En esta sección, se debe ingresar lo que es el número y ubicación de la misma. Además, esta clase contiene lo que son la cola de clientes, la pila del cajero y una lista de secciones.

Inicialmente se hacen los constructores, destructores, setters y getters de sus atributos. Siendo esto así, también contiene los respectivos to strings, los cuales son:

toString de la sucursal:

```
65  string Sucursal::toString() {
66      stringstream x;
67
68      x << "El ID es: " << id << "\n";
69
70      x << "La ubicacion es: " << ubicacion << "\n";
71
72      if (cajero == nullptr) {
73          x << "No hay cajero \n";
74      } else {
75          x << "Cajero: " << cajero->toString();
76      }
77
78      if (clientes == nullptr) {
79          x << "No hay clientes \n";
80      } else {
81          x << "Clientes: " << clientes->toString();
82      }
83
84      if (secciones->empty()) {
85          x << "No hay secciones \n";
86      } else {
87          x << toStringSecciones(*secciones);
88      }
89
90      return x.str();
91  }
```

En este toString, se imprimen los atributos de la clase correspondiente, además también cuenta con las siguientes verificaciones: el cajero, los clientes y las secciones, en donde se revisa que no estén vacíos.

Nodo sucursal

En esta clase se contienen los nodos de la lista de sucursales, la cual tiene como atributo el nodo siguiente, estando en una lista simple. Tiene su constructor, destructor, setters y getters de los atributos.

Sección

Esta clase se contiene en la clase sucursal. La clase sección contiene lo que es una lista de productos, y además tiene como atributos el nombre y número de sección. Contiene su constructor, destructor, setters y getters de los atributos. También contiene los respectivos toStrings.

```
33  string Seccion::toString() {
34      stringstream x;
35
36      x << "\tSeccion #" << numero << "\n";
37
38      x << "Nombre: " << nombre << "\n";
39
40      x << productos->toString();
41
42      return x.str();
43  }
```

En este toString como tal, se llama al toString de la lista de productos, así que además de mostrar la información de sus atributos, también muestra el de la lista.

Pila producto

La pila de productos contiene los métodos necesarios para agregar, hacer pop, obtener el tamaño de la pila, obtener el top de la pila y verificar que esté vacía. Estas pilas se agregan dentro de la lista de productos.

Nodo producto

En esta clase se contienen los nodos de la lista de productos, la cual tiene como atributo el nodo siguiente, anterior, y además contiene el producto de la pila

de productos, siendo una lista doblemente enlazada. Tiene su constructor, destructor, setters y getters de los atributos.

Lista producto

En esta lista se contienen los nodos con los respectivas pilas, además, contiene varios métodos, entre ellos insertar en la cabeza (o el primer nodo), el producto En este método, se crea un nuevo nodo del producto correspondiente, en el inicio de la lista.

```
67  void ListaProducto::insertarPrimero(PilaProducto* dato) {
68      cabeza = new NodoProducto(nullptr, dato, cabeza);
69  }
```

Se tiene un método el cual inserta en el ultimo nodo, un producto. Primero se verifica si el primer nodo está vacío, siendo así, se le inserta el producto. Caso contrario, se verifica que el que está a la par del primer nodo no está vacío, siendo así, pasará al siguiente nodo hasta encontrar uno sin datos, y a este se le insertará el producto.

```
71  void ListaProducto::insertarUltimo(PilaProducto* dato) {
72      actual = cabeza;
73
74      if (cabeza == nullptr) {
75          cabeza = new NodoProducto(nullptr, dato, cabeza);
76      } else {
77          actual = cabeza;
78          while (actual->getSig() != nullptr) {
79              actual = actual->getSig();
80          }
81          actual->setSig(new NodoProducto(actual, dato, nullptr));
82      }
83  }
```

Este método, lo que permite realizar es buscar un producto en especifico para eliminarlo, por medio de un string. Se recorren los nodos hasta encontrar el dato que se busca.

```
35  bool ListaProducto::eliminarTop(string producto) {
36      NodoProducto * Borrar = cabeza;
37      NodoProducto * Anterior = nullptr;
38
39      bool tempBorrado = false;
40
41      if (Borrar == nullptr) {//PRIMER NODO
42          return tempBorrado;
43      }
44
45      if (Borrar->getDato()->getTop() == producto) {
46          cabeza = Borrar->getSig();
47          delete Borrar;
48          Borrar = nullptr;
49          tempBorrado = true;
50      } else {
51          while ((Borrar != nullptr) && (Borrar->getDato()->getTop() != producto)) {
52              Anterior = Borrar;
53              Borrar = Borrar->getSig();
54          }
55          if (Borrar == nullptr) {
56              tempBorrado = false;
57          } else {
58              Anterior->setSig(Borrar->getSig());
59              delete Borrar;
60              Borrar = nullptr;
61              tempBorrado = true;
62          }
63      }
64      return tempBorrado;
65  }
```

En este método lo que se hace es un contador que permite devolver la cantidad de nodos que se contenga.

```
124 int ListaProducto::getTam() {
125     NodoProducto *aux = cabeza;
126     int contador = 0;
127
128     while (aux != nullptr) {
129         contador++;
130         aux = aux->getSig();
131     }
132     return contador;
133 }
```

Verifica cuantas pilas se encuentran vacías y devuelve la cantidad de estas. Mientras el nodo tenga un dato, y ese dato tenga una pila vacía, el contador suma.

```
135  int ListaProducto::cantPilasVacias() {
136      NodoProducto *aux = cabeza;
137      int contador = 0;
138
139      while (aux != nullptr) {
140          if (aux->getDato()->pilaVacia() == true) {
141              contador++;
142          }
143          aux = aux->getSig();
144      }
145      return contador;
146  }
```

Cola cliente

La cola cliente, contiene los clientes que contiene una pila de productos. Utiliza la biblioteca <queue> para manipular los clientes.

Pila cajero

La pila de cajero (es una pila de productos) contiene los métodos necesarios para agregar, hacer pop, obtener el tamaño de la pila, obtener el top de la pila y verificar que esté vacía.

Pila carrito

Esta pila la manipula el usuario para la compra de los productos. Tiene los mismos métodos de las pilas anteriores.

Controladora

Menú administrador

En esta clase se desarrollan los métodos necesarios para la función de la ferretería. Se divide en varios menús, empezando por el menú de administrador o cliente.

El menú administrador contiene lo siguiente:

En este menú, el administrador inserta una sucursal, ingresando el número y la ubicación correspondiente, edita sus atributos, muestra las sucursales que hay; y las elimina.

```
159
160             cout << "\t\t\tMenu del administrador\n";
161             cout << "Digite la opcion que desee ejecutar\n";
162             cout << " 1 - Insertar una nueva sucursal\n";
163             cout << " 2 - Editar sucursal\n";
164             cout << " 3 - Ver Sucursal Especifica\n";
165             cout << " 4 - Ver Todas las Sucursales\n";
166             cout << " 5 - Eliminar sucursal especifica\n";
167             cout << "77 - SALIR \n";
168             cout << "Opcion:  ";
169             getline(cin, opc);
```

Al ingresar en la opción de editar sucursal, se despliega otro menú que permite manipular las sucursales.

```
    system("clear");

    cout << "\t\t\tMenu editar Sucursal\n";
    cout << "Digite la opcion que desee ejecutar
    cout << " 1 - Editar ID\n";
    cout << " 2 - Editar Ubicacion\n";
    cout << " 3 - Editar Cajero\n";
    cout << " 4 - Editar ColaClientes\n";
    cout << " 5 - Editar Lista de Secciones\n";
    cout << "77 - SALIR \n";
    cout << "Opcion:  ";
    getline(cin, opc);

    try {
```

Al ingresar en la opción de editar cajero, ubicación, id, cola clientes, se despliega otros menús que permite agregar o eliminar cada atributo.

Menú cliente

En este menú el usuario puede comprar el producto, en la sucursal y sección que quiera.

Este método permite obtener un producto de la lista de productos. Se realiza un ciclo en donde, se verifica que en el nodo se encuentre un dato, siendo así, se verifica que el top sea el producto que el usuario desea. Luego se obtiene la cantidad de productos deseados. El contador permite saber el numero de productos que se agregó al carrito.

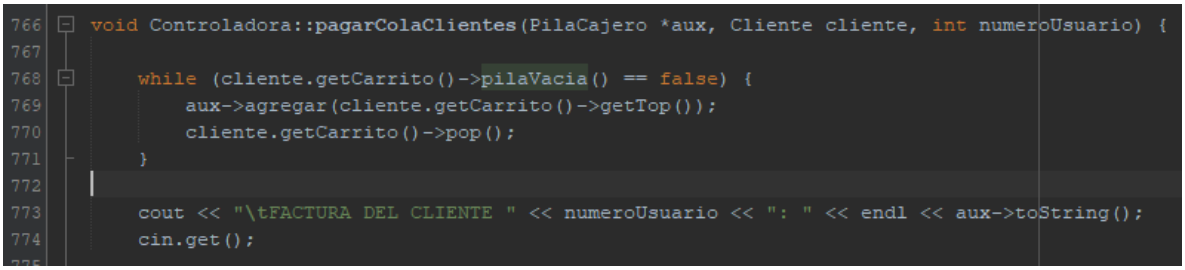
```
730  void Controladora::productoEspecifico(string producto, ListaProducto *lista, int cantProducto) {
731      NodoProducto *cabeza = lista->getCabeza();
732      string productoCarrito = "";
733      int contador = 0;
734      while (cabeza != nullptr) {
735
736          if (cabeza->getDato()->getTop() == producto) {
737              for (int i = 0; i < cantProducto; i++) {
738                  if (cabeza->getDato()->pilaVacia() == false) {
739                      productoCarrito = cabeza->getDato()->getTop();
740                      cabeza->getDato()->Pop();
741                      carritoCliente->agregar(productoCarrito);
742                      contador++;
743                  }
744              }
745              cout << productoCarrito << " se agrego al carrito " << contador << " veces" << endl;
746              cin.get();
747          }
748          cabeza = cabeza->getSig();
749
750      }
```

Otro método importante es el de pagar. Mientras la cola de clientes de la sucursal especifica tenga clientes, se llama al método de pagar cola clientes, que permite obtener los productos del carrito del cliente (no del usuario), y de esta forma generar su factura. Una vez el cliente paga el producto, se elimina de la cola, como se puede ver en la línea 758. Al terminar, la pila de productos (el cajero) se limpia para generar otra compra y el cliente paga.

```
753  void Controladora::pagar(NodoSucursal *aux) {
754      int contador = 1;
755      PilaCajero *cajero = aux->getDato()->getCajero();
756      while (aux->getDato()->getClientes()->colaVacia() == false) {
757          pagarColaClientes(cajero, aux->getDato()->getClientes()->getTop(), contador);
758          aux->getDato()->getClientes()->pop();
759          limpiarCajero(cajero);
760          contador++;
761      }
762      pagarUsuario(cajero);
763      limpiarCajero(cajero);
764  }
```

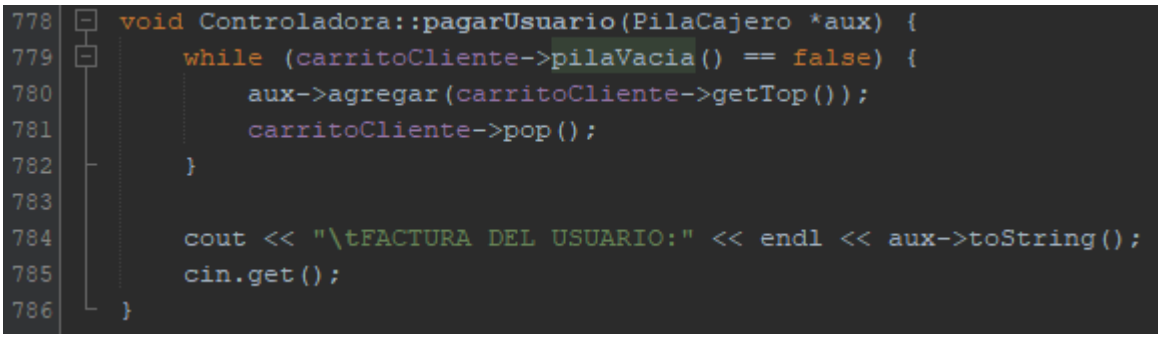
Este método se llama en el método de anterior, el cual permite al cliente pagar. Primero, mientras la pila del carrito del cliente tenga productos, al cajero se

le agrega los productos del carrito, y posteriormente eliminándolo del carrito. Al final se genera la factura.



```
766  void Controladora::pagarColaClientes(PilaCajero *aux, Cliente cliente, int numeroUsuario) {
767
768      while (cliente.getCarrito()->pilaVacia() == false) {
769          aux->agregar(cliente.getCarrito()->getTop());
770          cliente.getCarrito()->pop();
771      }
772
773      cout << "\tFACTURA DEL CLIENTE " << numeroUsuario << ": " << endl << aux->toString();
774      cin.get();
```

Este método permite al usuario pagar. Mientras el carrito contenga los productos, estos se agregan al cajero, y se procede a eliminarlo de la pila del carrito. Se genera la factura del usuario.



```
778  void Controladora::pagarUsuario(PilaCajero *aux) {
779      while (carritoCliente->pilaVacia() == false) {
780          aux->agregar(carritoCliente->getTop());
781          carritoCliente->pop();
782      }
783
784      cout << "\tFACTURA DEL USUARIO:" << endl << aux->toString();
785      cin.get();
786  }
```