Bing

Hola, este es Copilot. Estoy aquí para ayudarte con tu proyecto de programación. 😊

En base a este documento, el pseudocódigo con OOP para resolver este proyecto en C++ podría ser algo así:

```
// Definir una clase Producto que representa un producto del supermercado
class Producto {
  // Atributos privados: id, nombre, precio, stock
  private:
    int id;
    string nombre;
    float precio;
    int stock;

  // Métodos públicos: constructor, getters, setters, mostrar
  public:
    // Constructor que recibe los valores de los atributos
    Producto(int id, string nombre, float precio, int stock) {
      this->id = id;
      this->nombre = nombre;
      this->precio = precio;
      this->stock = stock;
    }

    // Getters y setters para acceder y modificar los atributos
    int getId() {
      return id;
    }

    void setId(int id) {
      this->id = id;
    }

    string getNombre() {
      return nombre;
    }

    void setNombre(string nombre) {
      this->nombre = nombre;
    }

    float getPrecio() {
      return precio;
    }

    void setPrecio(float precio) {
      this->precio = precio;
    }

    int getStock() {
      return stock;
    }

    void setStock(int stock) {
      this->stock = stock;
    }

    // Método para mostrar los datos del producto
    void mostrar() {
      cout << "ID: " << id << endl;
      cout << "Nombre: " << nombre << endl;
      cout << "Precio: " << precio << endl;
      cout << "Stock: " << stock << endl;
    }
}

// Definir una clase Carrito que representa el carrito de compras de un cliente
class Carrito {
  // Atributos privados: pila de productos, cantidad de productos, total a pagar
  private:
    stack<Producto> productos;
    int cantidad;
    float total;

  // Métodos públicos: constructor, agregarProducto, quitarProducto, vaciar, mostrar, getTotal
```

```
public:
  // Constructor que inicializa los atributos
  Carrito() {
    cantidad = 0;
    total = 0;
  }

  // Método para agregar un producto al carrito, verificando las restricciones
  void agregarProducto(Producto p) {
    // Si el carrito está lleno, mostrar un mensaje de error
    if (cantidad == 30) {
      cout << "El carrito está lleno, no se puede agregar más productos." << endl;
    }
    // Si el producto no tiene stock, mostrar un mensaje de error
    else if (p.getStock() == 0) {
      cout << "El producto " << p.getNombre() << " no tiene stock, no se puede agregar." << endl;
    }
    // Si el producto ya está en el carrito y se ha alcanzado el límite de 5 por tipo, mostrar un mensaje de error
    else if (estaEnCarrito(p) && contarProducto(p) == 5) {
      cout << "Ya se ha agregado el máximo de 5 productos de tipo " << p.getNombre() << ", no se puede agregar más." << endl;
    }
    // Si no hay ningún problema, agregar el producto al carrito, actualizar la cantidad, el total y el stock
    else {
      productos.push(p);
      cantidad++;
      total += p.getPrecio();
      p.setStock(p.getStock() - 1);
      cout << "Se ha agregado el producto " << p.getNombre() << " al carrito." << endl;
    }
  }

  // Método para quitar el último producto agregado al carrito, si hay alguno
  void quitarProducto() {
    // Si el carrito está vacío, mostrar un mensaje de error
    if (cantidad == 0) {
      cout << "El carrito está vacío, no se puede quitar ningún producto." << endl;
    }
    // Si no, quitar el producto de la pila, actualizar la cantidad, el total y el stock
    else {
      Producto p = productos.top();
      productos.pop();
      cantidad--;
      total -= p.getPrecio();
      p.setStock(p.getStock() + 1);
      cout << "Se ha quitado el producto " << p.getNombre() << " del carrito." << endl;
    }
  }

  // Método para vaciar el carrito, devolviendo los productos al stock
  void vaciar() {
    // Mientras el carrito no esté vacío, quitar cada producto
    while (cantidad > 0) {
      quitarProducto();
    }
    // Mostrar un mensaje de confirmación
    cout << "Se ha vaciado el carrito." << endl;
  }

  // Método para mostrar el contenido del carrito
  void mostrar() {
    // Si el carrito está vacío, mostrar un mensaje
    if (cantidad == 0) {
      cout << "El carrito está vacío." << endl;
    }
    // Si no, mostrar los productos en orden inverso al que se agregaron, usando una pila auxiliar
    else {
      cout << "El carrito contiene los siguientes productos:" << endl;
      stack<Producto> aux;
      // Pasar los productos del carrito a la pila auxiliar, mostrando cada uno
      while (!productos.empty()) {
        Producto p = productos.top();
        productos.pop();
        aux.push(p);
        p.mostrar();
        cout << "----------------------" << endl;
      }
      // Devolver los productos de la pila auxiliar al carrito, manteniendo el orden
      while (!aux.empty()) {
        Producto p = aux.top();
        aux.pop();
        productos.push(p);
```

```
      }
    }
  }

  // Método para obtener el total a pagar del carrito
  float getTotal() {
    return total;
  }

  // Método auxiliar para verificar si un producto está en el carrito, usando una pila auxiliar
  bool estaEnCarrito(Producto p) {
    bool encontrado = false;
    stack<Producto> aux;
    // Buscar el producto en el carrito, pasando los productos a la pila auxiliar
    while (!productos.empty() && !encontrado) {
      Producto q = productos.top();
      productos.pop();
      aux.push(q);
      // Si el producto tiene el mismo id que el buscado, cambiar el valor de encontrado a true
      if (q.getId() == p.getId()) {
        encontrado = true;
      }
    }
    // Devolver los productos de la pila auxiliar al carrito, manteniendo el orden
    while (!aux.empty()) {
      Producto q = aux.top();
      aux.pop();
      productos.push(q);
    }
    // Retornar el valor de encontrado
    return encontrado;
  }

  // Método auxiliar para contar cuántas veces aparece un producto en el carrito, usando una pila auxiliar
  int contarProducto(Producto p) {
    int contador = 0;
    stack<Producto> aux;
    // Contar el producto en el carrito, pasando los productos a la pila auxiliar
    while (!productos.empty()) {
      Producto q = productos.top();
      productos.pop();
      aux.push(q);
      // Si el producto tiene el mismo id que el buscado, incrementar el contador
      if (q.getId() == p.getId()) {
        contador++;
      }
    }
    // Devolver los productos de la pila auxiliar al carrito, manteniendo el orden
    while (!aux.empty()) {
      Producto q = aux.top();
      aux.pop();
      productos.push(q);
    }
    // Retornar el valor del contador
    return contador;
  }
}

// Definir una clase Cliente que representa un cliente del supermercado
class Cliente {
  // Atributos privados: cedula, nombre, apellido, telefono, carrito, tiempo
  private:
    int cedula;
    string nombre;
    string apellido;
    string telefono;
    Carrito carrito;
    int tiempo;

  // Métodos públicos: constructor, getters, setters, mostrar, comprar, generarTicket
  public:
    // Constructor que recibe los valores de los atributos
    Cliente(int cedula, string nombre, string apellido, string telefono) {
      this->cedula = cedula;
      this->nombre = nombre;
      this->apellido = apellido;
      this->telefono = telefono;
      // Inicializar el carrito vacío
      carrito = Carrito();
      // Generar un tiempo aleatorio entre 1 y 10 minutos
      tiempo = rand() % 10 + 1;
```

```
}

// Getters y setters para acceder y modificar los atributos
int getCedula() {
  return cedula;
}

void setCedula(int cedula) {
  this->cedula = cedula;
}

string getNombre() {
  return nombre;
}

void setNombre(string nombre) {
  this->nombre = nombre;
}

string getApellido() {
  return apellido;
}

void setApellido(string apellido) {
  this->apellido = apellido;
}

string getTelefono() {
  return telefono;
}

void setTelefono(string telefono) {
  this->telefono = telefono;
}

Carrito getCarrito() {
  return carrito;
}

void setCarrito(Carrito
```