MANUAL TÉCNICO

PRESENTACIÓN

El siguiente manual se ha desarrollado con la finalidad de dar a conocer la información necesario para realizar el mantenimiento y exploración del sistema Cobra Kai el cual consta de un sistema de autenticación y un apartado para agregar un producto de manera individual, carga masiva de archivos, generar reportes en html y salir. Realizado en el lenguaje JAVA aplicando los conceptos vistos en clase.

```
package org.example;

import org.example.security.UserAuth;
import org.example.ui.Menu;



public class Main {

   public static String PATH_EXPORT =
"C:\\Users\\ajolo\\OneDrive\\Documentos\\NetBeansProjects\\proyecto1";
   public static String PATH_IMPORT =
"C:\\Users\\ajolo\\OneDrive\\Documentos\\NetBeansProjects\\proyecto1\\products.txt";

   public static void main(String[] args) {

      UserAuth userAuth = new UserAuth();
      userAuth.login();

      Menu menu = new Menu();
      menu.displayMenu();


   }
}
```

Esta parte del código en Java se encarga de inicializar y ejecutar una aplicación que maneja autenticación de usuarios y presenta un menú de opciones.

- **Paquete**: El código pertenece al paquete org.example.
- **Importaciones**: Se importan las clases UserAuth y Menu de los paquetes org.example.security y org.example.ui respectivamente.
- **Instancia y Autenticación de Usuario**: Se crea una instancia de UserAuth y se llama al método login para autenticar al usuario.
- **Instancia y Presentación del Menú**: Se crea una instancia de Menu y se llama al método displayMenu para mostrar el menú de la aplicación.

```
package org.example.security;
import java.util.Scanner;
public class UserAuth {

  private  String username;
  private  String password;
  private final String CARNET = "201930494";

  public UserAuth() {
    this.username = "sensei_"+CARNET;
    this.password = "ipc1_"+CARNET;
  }

  public boolean authenticate(String username, String password) {
    return this.username.equals(username) && this.password.equals(password);
  }

  public boolean login() {
    Scanner scanner = new Scanner(System.in);
    System.out.println("=== SISTEMA DE AUTENTICACION COBRA KAI ===");
    System.out.println("=========================================");

    while (true) {
      System.out.print("Ingrese usuario: ");
      String username = scanner.nextLine();

      System.out.print("Ingrese contrasena: ");
      String password = scanner.nextLine();

      if (authenticate(username, password)) {
        System.out.println("Acceso concedido! Bienvenido al sistema, Sensei.");
        return true;
      } else {
        System.out.println("Error: Usuario o contrasena incorrectos. Intente
nuevamente.");
      }
    }
  }
}
```

Se encarga de la autenticación de usuarios en el sistema "SISTEMA DE AUTENTICACIÓN COBRA KAI". Define credenciales predeterminadas, solicita al usuario sus credenciales y las verifica en un bucle hasta que se ingrese la información correcta. Una vez autenticado, da la bienvenida al usuario.

```
package org.example.ui;

import org.example.Main;
import org.example.model.Inventory;
import org.example.model.Sale;
import org.example.model.Product;
import org.example.utils.HTMLGenerator;

import java.io.File;
import java.util.Scanner;

public class Menu {
    private final Inventory inventory;
    private final Sale[] salesHistory;
    private int saleCount;
    private final Scanner scanner;

    public Menu() {
        this.inventory = new Inventory();
        this.salesHistory = new Sale[100];
        this.saleCount = 0;
        this.scanner = new Scanner(System.in);
    }

    public void displayMenu() {
        boolean running = true;

        while (running) {
            System.out.println("\n=== MENU PRINCIPAL - COBRA KAI ===");
            System.out.println("1. Agregar producto individual");
            System.out.println("2. Carga masiva de productos");
            System.out.println("3. Realizar venta");
            System.out.println("4. Generar reporte Top 5 productos");
            System.out.println("5. Generar reporte historico de ventas");
            System.out.println("6. Salir");

            System.out.print("Seleccione una opcion: ");
            int option = scanner.nextInt();
            scanner.nextLine();

            switch (option) {
                case 1:
                    addProduct();
                    break;
```

```
            case 2:
                loadProduct();
                break;
            case 3:
                performSale();
                break;
            case 4:
                generateTop5Report();
                break;
            case 5:
                generateSalesHistoryReport();
                break;
            case 6:
                System.out.println("Saliendo del sistema. Gracias, Sensei!");
                running = false;
                break;
            default:
                System.out.println("Opción no válida, intente nuevamente.");
        }
    }
}

private void addProduct() {
    System.out.print("Ingrese el nombre del producto: ");
    String name = scanner.nextLine();

    System.out.print("Ingrese el precio del producto: ");
    double price = scanner.nextDouble();

    if (inventory.addProduct(name, price)) {
        System.out.println("Producto agregado exitosamente.");
    } else {
        System.out.println("Error: El producto ya existe o el precio no es valido.");
    }
}

private void performSale() {
    System.out.print("Nombre del cliente: ");
    String customerName = scanner.nextLine();

    System.out.print("NIT del cliente (C/F si no aplica): ");
    String customerNIT = scanner.nextLine();

    Sale sale = new Sale(customerName, customerNIT, 10);
```

```
    boolean buying = true;

    while (buying) {
      System.out.println("\n--- Productos Disponibles ---");
      Product[] products = inventory.getAllProducts();
      for (int i = 0; i < products.length; i++) {
        System.out.println((i + 1) + ". " + products[i].getName() + " - Q" +
products[i].getPrice());
      }

      System.out.print("Seleccione el número del producto (0 para finalizar): ");
      int productOption = scanner.nextInt();
      if (productOption == 0) {
        buying = false;
        break;
      }

      System.out.print("Ingrese la cantidad a comprar: ");
      int quantity = scanner.nextInt();
      scanner.nextLine();

      sale.addProduct(products[productOption - 1], quantity);
    }

    salesHistory[saleCount++] = sale;
    System.out.println("\nFactura:");
    sale.printInvoice();

    String pathExport = Main.PATH_EXPORT+ File.separator+ "factura_venta_" + saleCount
+ ".html";
    HTMLGenerator.generateInvoiceHTML(sale, pathExport);
  }

  private void generateTop5Report() {
    Product[] products = inventory.getAllProducts();
    String pathExport = Main.PATH_EXPORT+ File.separator +"reporte_top5.html";
    HTMLGenerator.generateTop5ProductsReport(products, pathExport);
  }

  private void generateSalesHistoryReport() {

    String pathExport = Main.PATH_EXPORT+ File.separator +"historial_ventas.html";
    HTMLGenerator.generateSalesHistoryReport(salesHistory, pathExport);
  }
```

```
    private void loadProduct() {
        System.out.print("Ingrese la ruta del archivo: ");
        String filePath = scanner.nextLine();
        if(filePath.isEmpty()) {
            filePath = Main.PATH_IMPORT;
        }
        inventory.loadProductFromFile(filePath);
    }

}
```

Se encarga de gestionar un sistema de inventario y ventas mediante un menú interactivo. Permite agregar productos individualmente o en masa, realizar ventas, y generar reportes en formato HTML. Utiliza clases para manejar inventarios, ventas y productos, así como utilidades para generar los reportes.

```
package org.example.model;

public class Product {
   private String name;
   private double price;
   private int quantitySold;

   public Product(String name, double price) {
      this.name = name;
      this.price = price;
      this.quantitySold = 0;
   }

   public String getName() {
      return name;
   }

   public double getPrice() {
      return price;
   }

   public int getQuantitySold() {
      return quantitySold;
   }

   public void incrementQuantitySold(int quantity) {
      this.quantitySold += quantity;
   }

   public void setPrice(double price) {
      this.price = price;
   }
}
```

Este código define la clase Product, que es fundamental para gestionar los productos en un sistema de inventario y ventas. La clase incluye atributos para el nombre, precio y cantidad vendida del producto, y proporciona métodos para acceder y modificar estos atributos. Además, incluye un método para incrementar la cantidad vendida.

```
package org.example.model;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Inventory {
    private Product[] products;
    private int size;
    private final int MAX_PRODUCTS = 100;

    public Inventory() {
        products = new Product[MAX_PRODUCTS];
        size = 0;
    }

    public boolean addProduct(String name, double price) {
        if (price <= 0 || productExists(name)) {
            return false;
        }

        if (size < MAX_PRODUCTS) {
            products[size++] = new Product(name, price);
            return true;
        } else {
            System.out.println("Error: El inventario está lleno.");
            return false;
        }
    }

    public boolean productExists(String name) {
        for (int i = 0; i < size; i++) {
            if (products[i].getName().equalsIgnoreCase(name)) {
                return true;
            }
        }
        return false;
    }

    public Product getProductByName(String name) {
        for (int i = 0; i < size; i++) {
            if (products[i].getName().equalsIgnoreCase(name)) {
                return products[i];
            }
```

```
    }
    return null;
  }

  public Product[] getAllProducts() {
    Product[] currentProducts = new Product[size];
    for (int i = 0; i < size; i++) {
      currentProducts[i] = products[i];
    }
    return currentProducts;
  }
  public void loadProductFromFile(String filePath) {
    if(filePath == null || filePath.isEmpty()) {
      System.out.println("!!! Debe enviar una ruta");
    }
    try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
      String line;
      boolean firstLine = true;

      while ((line = reader.readLine()) != null) {
        if (firstLine) {
          firstLine = false;
          continue;
        }

        String[] parts = line.split(";");
        if (parts.length == 2) {
          String productName = parts[0].trim();
          double price = Double.parseDouble(parts[1].trim());

          if (price > 0 && !productExists(productName)) {
            addProduct(productName, price);
          } else {
            System.out.println("Producto ignorado: " + productName + " (Precio inválido o
ya existe)");
          }
        } else {
          System.out.println("Línea malformada: " + line);
        }
      }
      System.out.println("Carga masiva completada.");
    } catch (IOException e) {
      System.out.println("Error al leer el archivo: " + e.getMessage());
    } catch (NumberFormatException e) {
```

```
            System.out.println("Error: Precio no válido en el archivo.");
        }
    }

}
```

Este código define la clase Inventory, que maneja el inventario de productos en el sistema. Proporciona métodos para agregar productos, verificar su existencia, obtener productos por nombre y cargar productos desde un archivo. También maneja las validaciones necesarias para asegurar la integridad de los datos del inventario.

```
package org.example.model;

public class Sale {
    private String customerName;
    private String customerNIT;
    private Product[] products;
    private int[] quantities;
    private int productCount;
    private double totalAmount;

    public Sale(String customerName, String customerNIT, int maxProducts) {
        this.customerName = customerName;
        this.customerNIT = customerNIT;
        this.products = new Product[maxProducts];
        this.quantities = new int[maxProducts];
        this.productCount = 0;
        this.totalAmount = 0;
    }

    public void addProduct(Product product, int quantity) {
        products[productCount] = product;
        quantities[productCount] = quantity;
        totalAmount += product.getPrice() * quantity;
        product.incrementQuantitySold(quantity);
        productCount++;
    }

    public double getTotalAmount() {
        return totalAmount;
    }

    public String getCustomerName() {
        return customerName;
    }

    public String getCustomerNIT() {
        return customerNIT;
    }

    public void printInvoice() {
        System.out.println("Factura de Venta");
        System.out.println("Cliente: " + customerName);
        System.out.println("NIT: " + customerNIT);
        System.out.println("Productos comprados:");
```

```
      for (int i = 0; i < productCount; i++) {
        System.out.println("- " + products[i].getName() + " x" + quantities[i] + " - Q" +
(products[i].getPrice() * quantities[i]));
      }
      System.out.println("Total: Q" + totalAmount);
    }
    public int getProductCount() {
      return productCount;
    }
    public Product[] getProducts() {
      return products;
    }

    public int[] getQuantities() {
      return quantities;
    }

}
```

Este código define la clase Sale, que gestiona las ventas en el sistema. Incluye métodos para agregar productos a una venta, calcular el monto total, y generar una factura detallada. La clase también permite obtener información sobre el cliente y los productos comprados, lo cual es fundamental para la gestión de ventas y generación de reportes.

```
package org.example.utils;

import org.example.model.*;

import java.io.FileWriter;
import java.io.IOException;

public class HTMLGenerator {

    public static void generateInvoiceHTML(Sale sale, String filePath) {
        try (FileWriter writer = new FileWriter(filePath)) {
            writer.write("<html><head>");
            writer.write("<title>Factura de Venta</title>");
            writer.write("<link rel='stylesheet'
href='https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css'>");
            writer.write("</head><body class='container mt-5'>");

            writer.write("<h1 class='text-center mb-4'>Factura de Venta - Cobra Kai</h1>");
            writer.write("<p><strong>Cliente:</strong> " + sale.getCustomerName() + "</p>");
            writer.write("<p><strong>NIT:</strong> " + sale.getCustomerNIT() + "</p>");

            writer.write("<h2 class='mt-4'>Productos Comprados</h2>");
            writer.write("<table class='table table-bordered table-striped'>");
            writer.write("<thead class='thead-
dark'><tr><th>Producto</th><th>Cantidad</th><th>Subtotal</th></tr></thead>");
            writer.write("<tbody>");

            for (int i = 0; i < sale.getProductCount(); i++) {
                Product product = sale.getProducts()[i];
                int quantity = sale.getQuantities()[i];
                double subtotal = product.getPrice() * quantity;

                writer.write("<tr>");
                writer.write("<td>" + product.getName() + "</td>");
                writer.write("<td>" + quantity + "</td>");
                writer.write("<td>Q" + String.format("%.2f", subtotal) + "</td>");
                writer.write("</tr>");
            }

            writer.write("</tbody></table>");
            writer.write("<h3 class='text-right mt-3'>Total: Q" + String.format("%.2f",
sale.getTotalAmount()) + "</h3>");
```

```
            writer.write("</body></html>");
            System.out.println("Factura generada exitosamente en: " + filePath);
        } catch (IOException e) {
            System.out.println("Error al generar la factura HTML: " + e.getMessage());
        }
    }

    public static void generateTop5ProductsReport(Product[] products, String filePath) {
        try (FileWriter writer = new FileWriter(filePath)) {
            writer.write("<html><head>");
            writer.write("<title>Top 5 Productos Más Vendidos</title>");
            writer.write("<link rel='stylesheet'
href='https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css'>");
            writer.write("</head><body class='container mt-5'>");

            writer.write("<h1 class='text-center mb-4'>Top 5 Productos Más Vendidos</h1>");
            writer.write("<table class='table table-bordered table-striped'>");
            writer.write("<thead class='thead-
dark'><tr><th>#</th><th>Producto</th><th>Cantidad Vendida</th></tr></thead>");
            writer.write("<tbody>");

            for (int i = 0; i < Math.min(5, products.length); i++) {
                writer.write("<tr>");
                writer.write("<td>" + (i + 1) + "</td>");
                writer.write("<td>" + products[i].getName() + "</td>");
                writer.write("<td>" + products[i].getQuantitySold() + "</td>");
                writer.write("</tr>");
            }

            writer.write("</tbody></table>");
            writer.write("</body></html>");
            System.out.println("Reporte Top 5 generado exitosamente en: " + filePath);
        } catch (IOException e) {
            System.out.println("Error al generar el reporte HTML: " + e.getMessage());
        }
    }

    public static void generateSalesHistoryReport(Sale[] sales, String filePath) {
        try (FileWriter writer = new FileWriter(filePath)) {
            writer.write("<html><head>");
            writer.write("<title>Historial de Ventas</title>");
            writer.write("<link rel='stylesheet'
href='https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css'>");
            writer.write("</head><body class='container mt-5'>");
```

```
        writer.write("<h1 class='text-center mb-4'>Historial de Ventas</h1>");
        writer.write("<table class='table table-bordered table-striped'>");
        writer.write("<thead class='thead-
dark'><tr><th>Cliente</th><th>NIT</th><th>Total</th></tr></thead>");
        writer.write("<tbody>");

        for (Sale sale : sales) {
          if (sale == null) continue;

          writer.write("<tr>");
          writer.write("<td>" + sale.getCustomerName() + "</td>");
          writer.write("<td>" + sale.getCustomerNIT() + "</td>");
          writer.write("<td>Q" + String.format("%.2f", sale.getTotalAmount()) + "</td>");
          writer.write("</tr>");
        }

        writer.write("</tbody></table>");
        writer.write("</body></html>");
        System.out.println("Historial de ventas generado exitosamente en: " + filePath);
      } catch (IOException e) {
        System.out.println("Error al generar el historial de ventas HTML: " + e.getMessage());
      }
    }
}
```

Este código define la clase HTMLGenerator, que incluye métodos para generar reportes en formato HTML para facturas de ventas, los 5 productos más vendidos y el historial de ventas. Utiliza FileWriter para escribir en archivos HTML, estructura los reportes con etiquetas HTML.