Carátula para entrega de prácticas	
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	ING. JORGE A HERNÁNDEZ LÓPEZ
Asignatura:	PROGRAMACIÓN ORIENTADA A OBJETOS
Grupo:	05
No de Práctica(s):	02
Integrante(s):	Hernández Chávez Samuel 320339147 Ovando Hernández Jonathan Daniel 321059145 Velazco Cancino Juan Antonio 321017149
No. de Equipo de cómputo empleado:	
No. de Lista o Brigada:	
Semestre:	2025-1
Fecha de entrega:	26 de agosto del 2024
Observaciones:	

CALIFICACIÓN: _____

INTRODUCCIÓN

En esta práctica se resolverán dos sencillos ejercicios los cuales consisten en modificar los códigos hechos en clase con el fin de hacerlos más eficientes así como practicar la utilización de colecciones en Java. Con esta práctica podremos asimilar adecuadamente las características y atributos del manejo de clases y métodos así como comenzar a desarrollar nuestra lógica de programación orientada a objetos.

DESARROLLO

Programa 1.- A partir del código realizado en la práctica, Registro.java, "Registro Bicis", modificarlo para conseguir que el sistema tenga la capacidad de Registrar a más de una persona y su vez pueda validar el NIP de cualquier persona registrada.

Primero se deben importar las librerías Scanner,HashMap y Map.

Clase ”Alumno”.

```
//Clase Alumno, lo usamos como estrucutra, pero al tene
class Alumno{
    String nombre;
    String cuenta;
    int nip;
    String serieBici;

    void mostrarDatos(){
        System.out.println("Nombre: "+this.nombre);
        System.out.println("Cuenta: "+cuenta);
        System.out.println("Serie Bici: "+serieBici);
    }
}
```

En esta clase tenemos como atributos strings como nombre,cuenta y la serie de la bici, como int tenemos el nip, el unico metodo de esta clase se llama mostrarDatos(), y su función es imprimir los datos de la clase Alumno, no recibe ningún parámetro y no tiene valores de retorno.

Clase ”Registro”.

Comenzamos creando un HashMap llamado usuarios el cual únicamente puede contener objetos de tipo Alumno, aquí es donde se guardaran todos los usuarios, elegimos trabajar con HashMap porque es rápido y eficiente en la búsqueda de información(clave, valor).

```
class Registro{

    //Mapa de Usuarios, declarado a nvl global para acc
    Map<String, Alumno > Usuarios = new HashMap<>();
```

Método “pedirDatos()”:

Se declara un objeto de la clase Scanner(sc) para poder leer los datos introducidos por el usuario, después creamos un objeto de tipo Alumno (alu) el cual nos servirá porque es el objeto al cual se le llenaran los datos, después verificamos el usuario, si existe se seguirán pidiendo los datos y al final se guardarán en el HashMap (clave, valor), y se imprimen los datos obtenido. Recordando que el usuario ya ha sido guardado en el mapa.

```
void pedirDatos(){
    Scanner sc;
    sc=new Scanner(System.in);
    Alumno alu=new Alumno();
    System.out.println("\nIngresa tu nombre:");
    alu.nombre=sc.nextLine();
    System.out.println("\nIngresa tu numero cuenta:");
    alu.cuenta=sc.nextLine();
    //Verificar que el no de cuenta ya este registrado en el mapa
    boolean c = verificarUsuario(alu);
    //Si lo esta se detiene
    if(c==false)return;
    System.out.println("\nDame la serie de tu bici:");
    alu.serieBici=sc.nextLine();
    System.out.println("\nIntroduce tu NIP:");
    alu.nip=sc.nextInt();
    //Primer registro , inserta al mapa
    Usuarios.put(alu.cuenta,alu);
    //Mostramos los datos
    System.out.println("Datos Registrados con exito!!!\nVerifique que todo este bien ");
    alu.mostrarDatos();
```

Método “verificarUsuario()”:

Lo que hacemos es comparar los números de cuenta del mapa con el proporcionado de esta forma se regresa un False o True y así continúa o termina la ejecución del registro y te dirige de nuevo al Menú.

```
//Funcion que verifica que el usuario no este ya registrado
boolean verificarUsuario(Alumno al){
    if (Usuarios.containsKey(al.cuenta)) {
        System.out.println("\nNumero de cuenta ya registrado ... ");
        return false;
    }else{
        return true;
    }
}
```

Método “leerNoCuenta()”:

Se encarga de verificar de quién es el número de cuenta ingresada, nos regresa una string.

```java
//Funcion para saber de quien vamos a verificar el nip
String leerNoCuenta(){
    Scanner sc = new Scanner(System.in);
    System.out.println("Ingrese numero de cuenta");
    String noCuenta = sc.nextLine();
    return noCuenta;
}
```

Método “validaNIP()”:

Recibe como parámetro el número de cuenta así sabrá identificar el alumno correcto, si no existe inmediatamente regresa al menú, prácticamente compara el NIP leído con el valor que tiene el alumno asociado a ese número de cuenta, de esta forma el alumno puede guardar o no la bici.

Método “principal()”:

```java
void validaNIP(String noCuenta){
    if(!Usuarios.containsKey(noCuenta)){
        System.out.println("\nUsuario no registrado\nRegresando a Menu");
        return;
    }

    Alumno al = new Alumno();
    al = Usuarios.get(noCuenta);

    Scanner sc=new Scanner(System.in);
    System.out.println("Introduce tu nip registrado:");
    int nipLeido=sc.nextInt();

    if(nipLeido== al.nip){
        System.out.println("Validacion exitosa, puedes guardar tu bici...");
    }else{
        System.out.println("Contrasenia incorrecta!!!");
    }
```

Es un ciclo para mostrar siempre el menú.

```java
//Main
public static void main(String[] asd){
    Registro reg=new Registro();
    while(true){
        reg.mostrarMenu();
    }
}
```

Método “mostrarMenu()”:

Prácticamente lo que haces es desplegar el menú con las opciones disponibles y para salir utilizamos el método System.exit(0) con el cual termina la ejecución.

```java
void mostrarMenu(){
    System.out.println("******** MENU *******");
    System.out.println("1) Registrar");
    System.out.println("2) Ingresar");
    System.out.println("3) Salir");

    System.out.println("=>");
    Scanner sc=new Scanner(System.in);
    int opc=sc.nextInt();


    switch(opc){
        case 1:
            pedirDatos();
            break;
        case 2:
            String noCuenta = leerNoCuenta();
            validaNIP(noCuenta);
            break;
        case 3:
            System.exit(0);
        default:
            System.out.println("\n\nOpcion no valida...");
    }

}
```

Programa 2.- Dado el código de Autenticacion.java, a completarlo para que se convierta en un sistema de Login (Autenticación) dado un conjunto de datos almacenados previamente.

Primero importamos la biblioteca java.util.Scanner, esta nos servirá para leer datos desde el teclado.

Clase ”Autenticacion”.

Contiene los métodos necesarios para gestionar la autenticación de usuarios. La clase tiene un atributo “datos“, que es un arreglo de objetos “Usuario“ que almacenará los datos de los usuarios registrados.

Método “cargarDatos()”:

Este método se encarga de cargar los datos de los usuarios en el arreglo “datos“. Utiliza un bucle for para iterar 10 veces, lo que permite registrar 10 usuarios.

El scanner se utiliza para leer las entradas del usuario (nombre de usuario y contraseña).

Cada iteración del Arreglo de “usuario” crea un nuevo objeto Usuario y le asigna un nombre de usuario (login) y una contraseña (password) introducidos por el usuario.

Al final, imprime un mensaje que confirma que los datos han sido cargados.

```java
import java.util.Scanner;
//Scanner para leer datos

class Autenticacion{
    Usuario datos[];
    //Arreglo de datos
    void cargarDatos(){
        Scanner sc = new Scanner(System.in);

        datos=new Usuario[10];

        //Cargamos los datos via terminal
        for(int i = 0;i<10;i++){
            datos[i]=new Usuario();
            datos[i].login= sc.nextLine();
            datos[i].password= sc.nextLine();
        }

        System.out.println("\nDatos Cargados\n");
    }
```

Método “login()”:

Este método simula el proceso de login para un usuario. Pide al usuario que ingrese su nombre de usuario y contraseña, llama al método “verificar()” para comprobar si las credenciales proporcionadas coinciden con alguna de las almacenadas en “datos”. Si las credenciales son correctas, se imprime un mensaje de bienvenida.

```java
    //Simulamos el login
    void login(){
        Scanner sc;
        sc=new Scanner(System.in);

        Usuario user = new Usuario();

        System.out.println("Ingrese usuario");
        user.login = sc.nextLine();
        System.out.println("Ingrese Contrasenia");
        user.password = sc.nextLine();

        //Verificamos que los datos coincidan con los generados
        boolean c =verificar(user);

        if(c){
            System.out.println("Bienvenido!!\n\n");
        }
    }
```

Método “verificar(Usuario user)”:

Este método verifica si el nombre de usuario y la contraseña proporcionados coinciden con los registrados en el sistema. El Bucle for Itera a través de los 10 usuarios almacenados en datos. La primer condición if Compara el nombre de usuario (login) ingresado con los que están almacenados, en el segundo if si el nombre de usuario coincide, compara las contraseñas.

Finalmente si ambos coinciden, devuelve “true”, lo que significa que el usuario ha sido autenticado correctamente. Si no coinciden, imprime un mensaje de acceso denegado o de usuario no encontrado y devuelve “false”.

```
42      /*Uso de metodo equals ya que no es un dato primitivo*/
43      boolean verificar(Usuario user){
44          for (int i = 0; i < 10; i++) {
45              if(user.login.equals(datos[i].login)){
46                  if(user.password.equals(datos[i].password)){
47                      return true;
48                  }else{
49                      //Si la contraenia no coincide
50                      System.out.println("\nAcceso Denegado...");
51                      return false;
52                  }
53              }
54          }
55
56          //Si el usuario no es encontrado
57          System.out.println("\nUsuario no encontrado");
58          return false;
59      }
60
```

Método “main()”:

Este es el método principal del programa que ejecuta el mismo.Este crea un objeto de la clase “Autenticacion”, llama al método “cargarDatos()” para cargar los usuarios. El bucle Infinito ejecuta continuamente el menú para permitir al usuario realizar acciones como el login.

```
//Main
public static void main(String[] args) {
    Autenticacion au = new Autenticacion();
    au.cargarDatos();
    while (true) {
        au.menu();
    }
}
```

Método “menu()”:

Este método muestra las opciones que puede realizar el usuario(2), las cuales son:

1: Llamar al método login() para intentar iniciar sesión.

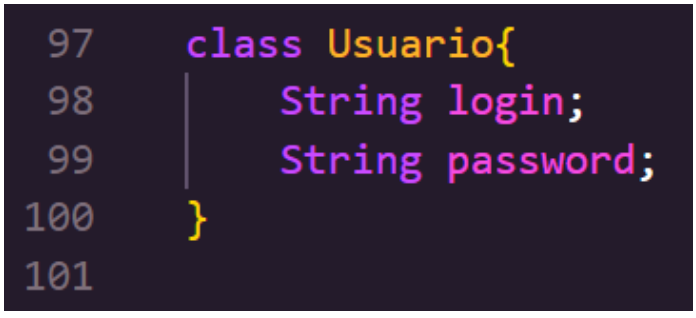
2: Terminar el programa con System.exit(0).

Lee la opción seleccionada por el usuario y ejecuta la acción correspondiente y si la opción ingresada no es válida, imprime un mensaje indicando que la opción no es válida.

```
71      void menu(){
72
73          System.out.println("******** MENU *******");
74          System.out.println("1) Ingresar");
75          System.out.println("2) Salir");
76
77          System.out.println("=>");
78          Scanner sc=new Scanner(System.in);
79
80          int opc=sc.nextInt();
81
82          switch(opc){
83              case 1:
84                  login();
85                  break;
86              case 2:
87                  System.exit(0);
88              default:
89                  System.out.println("\n\nOpcion no valida...");
90          }
91
92      }
93
```

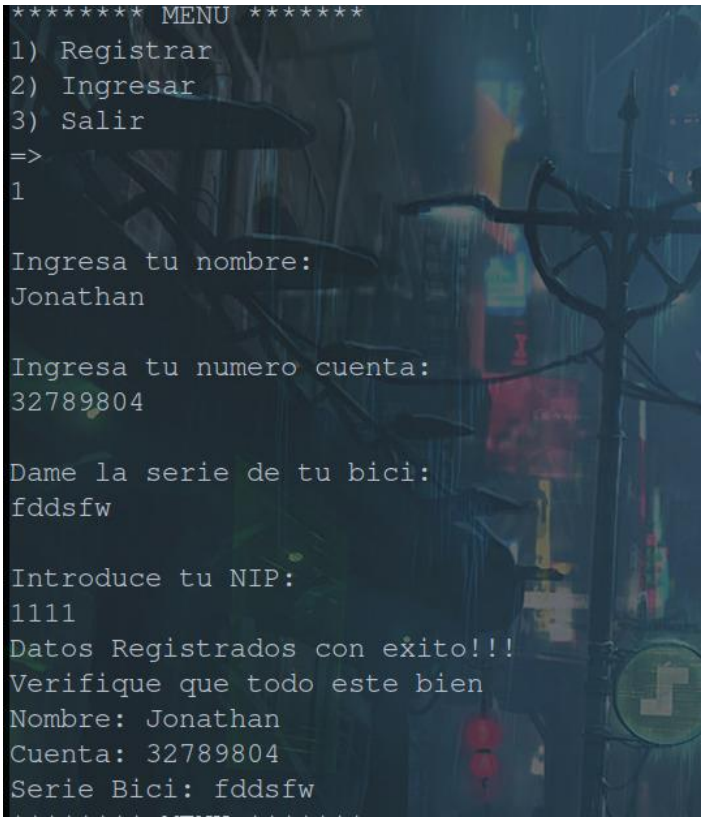
Clase ”Usuario”.

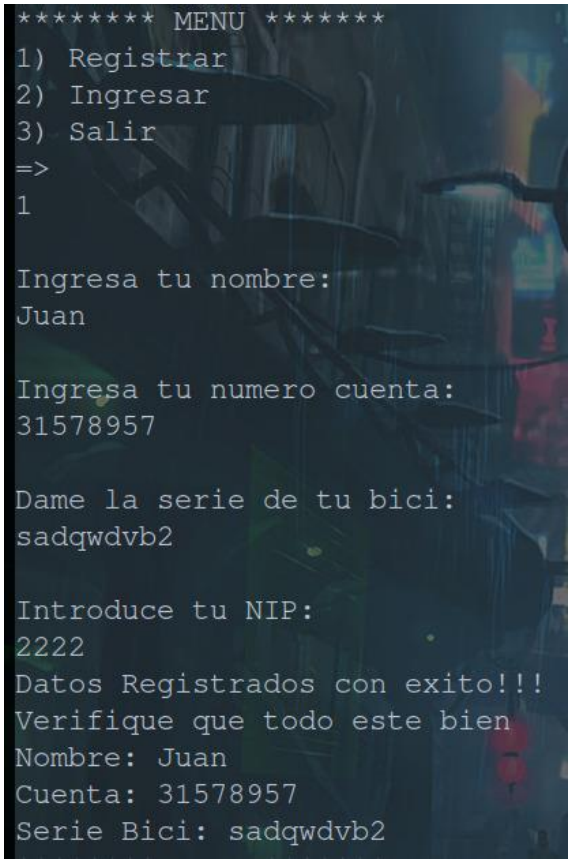
Esta clase se utiliza para representar a los usuarios que pueden registrarse y autenticarse en el sistema, contiene los atributos: “login” que se refiere al nombre de usuario y “password” que se refiere a la contraseña.

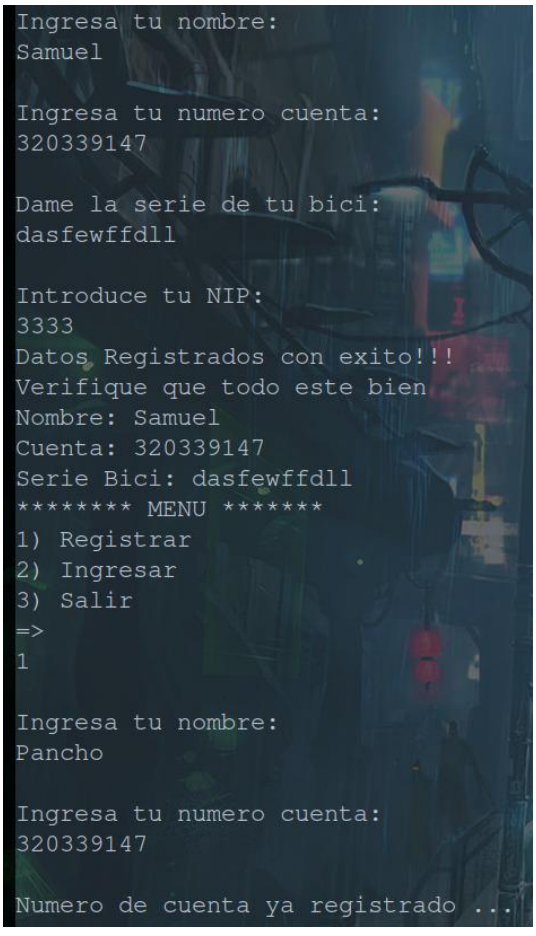


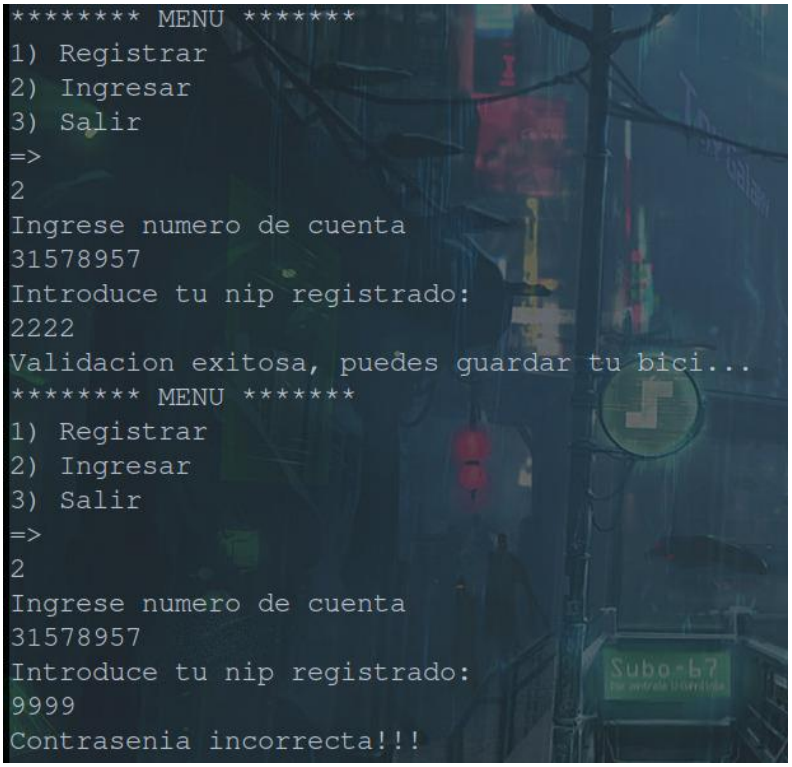
```
97      class Usuario{
98          String login;
99          String password;
100     }
101
```

Corrida del programa 1:

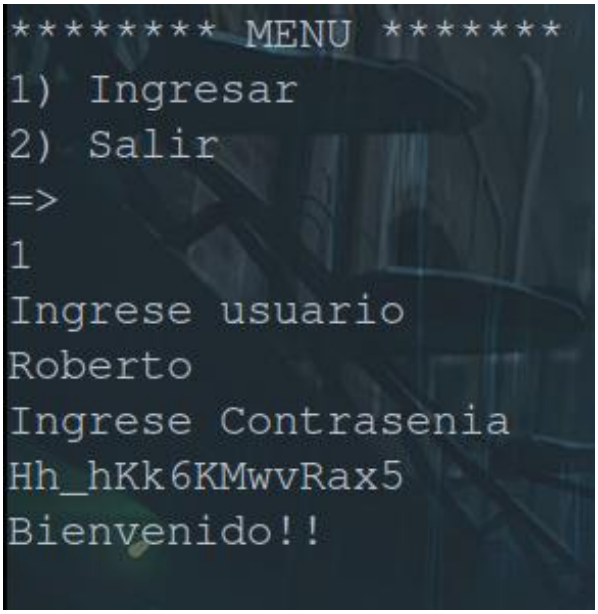


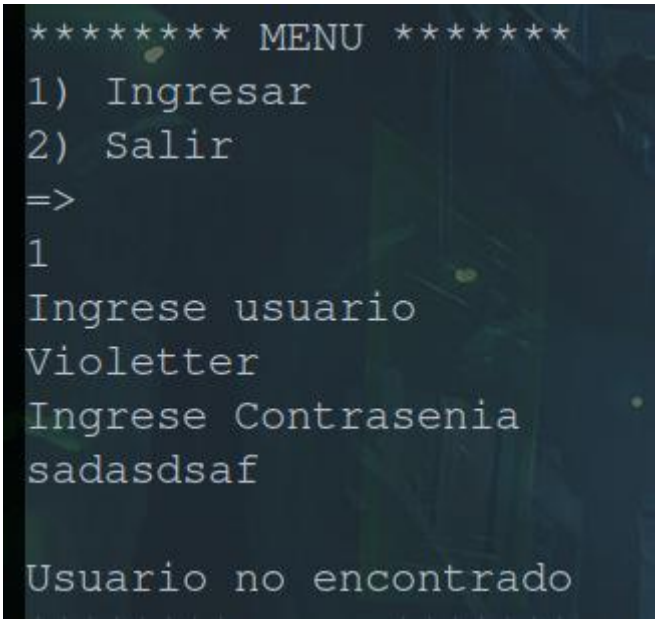


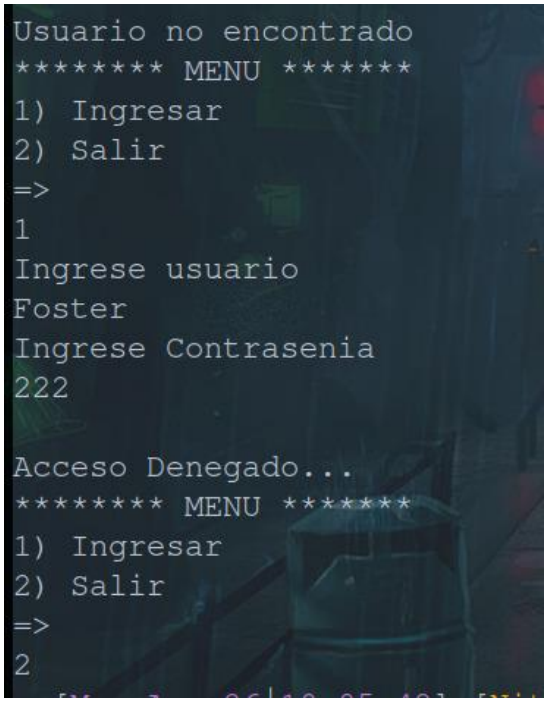




Corrida del programa 2:







CONCLUSIONES.

Hernández Chávez Samuel

Se me hizo tedioso tener que buscar los métodos de los contenedores dinámicos, como hashmap, map y set. Porque estoy acostumbrado de que en c++ para insertar y acceder es como si fuera un arreglo primitivo y los métodos usuales, los cuales ahorran demasiado tiempo, pero, al ser java se entiende, por que buscamos el identificar errores con mayor facilidad, y es como escribir ingles lo cual , en parte, logra ser más legible.

Por otro lado siempre los arreglos estáticos, son una herramienta muy útil para resolver programas que no involucre aumentar o disminuir el tamaño del array.

Ovando Hernández Jonathan Daniel

Yo personalmente gracias a la práctica aprendí el uso de las colecciones y como podemos utilizarlas para sacarle provecho a sus diferentes tipos de atributos, aún me cuesta trabajar con objetos y clases pero poco a poco voy familiarizándome con la sintaxis y más que nada con la POO, también aún no diferenció totalmente los diferentes tipos de Mapas pero espero ir conociendo sus ventajas y aprender a programarlos y utilizarlos en mis programas de forma más cotidiana.

Velazco Cancino Juan Antonio.

En esta práctica, logré entender más sobre las colecciones en Java, principalmente lo que puedo ver es su potencial para optimizar la gestión de datos en nuestros programas. Aunque el cambio de trabajar con estructuras estáticas a dinámicas me ha sido más complicado, esto especialmente en el manejo de objetos y la implementación de POO, considero que estos conceptos son importantes para desarrollar un código más flexible y sencillo.