Taller de herramientas de programación

Declaraciones de variables	Constantes	Asignaciones

boolean b;
int numero;
float decimal = 43.32e3f;
int contador = 0;
char c = 'a';
	static final tipo identificador = valor;
	static final **float** pi = 3.141592F;
		boolean javaEsGenial;
		javaEsGenial = *true*;
		int edad;
		edad = 22;
		String st1 = "Hola";
		String st3 = st1 + " " + st2; //
		st3 vale "Hola ¿cómo estás?"

numeroEntero = Integer.parseInt(input.nextLine());
numeroReal = Double.parseDouble(input.nextLine());

Parcial…
leer varias veces el enunciado, -revisar el zip varias veces, -lo primero que hay que hacer es cambiarle el nombre de al proyecto, -si un método no te sale haz otro método y no te quedes, -los enum pueden tener métodos
Observaciones de los parciales - mal:
recorrer más de una vez una lista
omitir las interfaces
getClass.name equis cosa para mostrar el nombr de la clase
no usar el instaOff
preguntar si el objeto era null antes de agregarlo

Leer - Leer Integer - Leer Double
input.nextLine(); - Integer.parseInt(input.nextLine()); - Double.parseDouble(input.nextLine());

Leer todo en MAYUSCULAS - .toUpperCase(); || EJEMPLO: VARIABLE = input.nextLine().toUpperCase();

Leer CHAR - genero = input.nextLine().charAt(0); // genero = input.nextLine().toUpperCase().charAt(0);

EQUALS
variable.equals(COMPARACION) || EJEMPLO: System.out.println("Estante " + j);

|| El signo de exclamación NiEGA la validación, el equals lo compara
|| Esto quiere decir *MIENTRAS nombre Libro SEA DISTINTO DE FIN HACER*

|| SCANNER ||
import java.util.Scanner;

public static Scanner input = new Scanner(System.in);

Siempre que uso un scanner siempre lo tengo q cerrar input(o nombre que puse).close , va a final de todo

Después del nombre de la variable

Si es el ingreso de teclado es para tipo string solamente coloco la palabra `teclado.nextLine();` después del nombre de la variable

Para numero enteros tengo q poner `Integer.parseInt(input.nextLine());` es sirve para convertir lo q viene en el teclado en un tipo de dato entero para asignar dentro de una variable entera , esto cambia si uso doublé , `Double.parseDouble(input.nextLine());` para un variable de ese estilo (ambos casos el interger y el doublé va con la primera letra mayúscula)

! es para negar

!= Es distinto

toUpperCase es para pasar todo a mayúscula (para que si el usuario pone en minúscula y yo lo necesito en mayúscula)

toLowerCase = para pasar todo a minúscula

equalsIgnoreCase = es para ignorar como puso si en mayúscula o en minuscula

ATRIBUTOS DE LA CLASE (STATIC)

Una variable estática (**static**) en **Java** es una variable que pertenece a la clase en que fue declarada y se inicializa solo una vez al inicio de la ejecución del programa, la característica principal de este tipo de variables es que se puede acceder directamente con el nombre de la clase sin necesidad de crear un objeto.

```java
//atributos de instancia(los que tendran los objetos creados
// a partir de esta clase
private int nroFactura;
private int codCliente;
private double importe;

// atributos de la clase(static)
private static double alicIva;
private static double alicIvaReducida;
private static int ultimaFactGenerada;

Factura(int codCliente, double importe) {
    this.nroFactura = Factura.ultimaFactGenerada+1;
    this.setCodCliente(codCliente);
    this.importe = importe;
    Factura.ultimaFactGenerada = this.nroFactura;

}
```

Podemos tener métodos staticos propios de la clase
Estos métodos son solos de la clase. No de la instancia .

```java
public static void actualizarIva(double iva) {
    Factura.alicIva = iva;
}

public static void actualizarIvaReducido(double iva) {
    Factura.alicIvaReducida = iva;
}
```

ArrayList(Colecciones de Objetos)

En Java, la forma más simple de trabajar con colecciones de objetos es utilizando la clase ***ArrayList*** que permite guardar en forma dinámica y secuencial un elemento detrás del otro, acceder a los mismos y eliminarlos cuando lo necesitemos.

Algunos de los métodos de los arraylist son

- **add**(Elemento) = se utiliza para agregar un elemento al final de la colección
- add(n, elemento) = Permite agregar un elemento en la posición señalada por n. Si esta posición está fuera del rango aceptable (determinado por la cantidad de elementos de la colección) se producirá un error de ejecución.
- **get** (n) = Permite obtener el elemento n de la colección
- **remove** (n) = permite eliminar un elemento de la colección
- remove(elemento) = Permite extraer el elemento recibido de la colección. A diferencia del otro remove que recibe la posición del elemento a extraer, como ya tiene el elemento devuelve verdadero o falso dependiendo de si se pudo o no extraer el elemento (devuelve falso cuando el elemento indicado no estaba insertado en la colección, cuando era ajeno a ella).

- **size** ()= nos devuelve el largo del array
- **isEmpty()** = Devuelve un valor booleano que indica si la colección está vacía. Es equivalente a preguntar si count() == 0.
- **contains()** = Devuelve true si esta lista sobre la que se invoca contiene todos los elementos de la colección pasada por parámetro. se busca el objeto en sí dentro del arraylist

```
12          System.out.println("Ingrese su edad: ");
13          numero = Integer.parseInt(input.nextLine());
14          System.out.println("La variable numero ahora vale: " + numero
15          //numero = numero*2;
16          //System.out.println("El doble de la variable numero ahora va
17          double division = numero / 7.0;
18          System.out.println("La variable division vale: " + division);
19
20
```

Javadoc | Declaración | Buscar | Progreso | Consola

```
<terminado> HolaMundo (1) [Aplicación Java] C:\Program Files\Java\jre1.8.0_281\bin\javaw.exe(15 de mar. de 2021 20:08:52)
Hola Curso A
La variable numero vale: 27
Ingrese su edad:
30
La variable numero ahora vale: 30
La variable division vale: 4.285714285714286
```

```
public Mascota buscarMascota(String nombreDeMascota) {
    // TODO A completar
    int i = 0;
    Mascota m = null;
    while(i < lMascotas.size() && m==null){
        if (lMascotas.get(i).getNombre().equals(nombreDeMascota)){
            m = lMascotas.get(i);
        }
        i++;
    }
    return m;
```

Tipo de operador	Operador	Descripción	Ejemplo
Aritmético	+	Suma	a+b
	-	Resta	a-b
	*	Multiplicación	a*b
	/	División	a/b
	%	Módulo	a%b
Relacional	>	Mayor que	a>b
	<	Menor que	a<b
	>=	Mayor o igual que	a>=b
	<=	Menor o igual que	a<=b
	!=	Diferente	a!=b
	==	Igual	a==b
Lógico	!	No	!a
	&&	Y	a&&b
	||	O	a||o
Manipulación de bits	~	Complemento	~a
	&	Y bit a bit	a&b
	|	O bit a bit	a|b
	^	O exclusivo bit a bit	a^b
	<<	Desplazamiento a la izda.	a<<b
	>>	Desplazamiento a la dcha.	a>>b
	>>>	Desplazmiento a la dcha. rellenando ceros	a>>>b
Asignación	=	Asignación	a=b
	++	Incremento y asignación	a++
	--	Decremento y asignación	a--
	+=	Suma y asignación	a+=b
	-=	Resta y asignación	a-=b
	*=	Multiplicación y asignación	a*=b
	/=	División y asignación	a/=b
	%=	Módulo y asignación	a%=b
	|=	O y asignación	a|=b
	&=	Y y asignación	a&=b
	^=	O exclusiva y asignación	a^=b
	<<=	Desplazamiento a la izda. y asignación	a<<=b
	>>=	Desplazamiento a la dcha. y asignación	a>>=b
	>>>=	Desplazamiento a la dcha. rellenando ceros y asignación	a>>>=b
Conversión de tipo (casting)	(tipo)	Convertir tipo	(char)b
Instancia	instanceof	¿Es instancia de clase)	a instanceof b

Condicionales

```
if(condición){
    …enunciados a ejecutar si se cumple la condición…
} else {
    …enunciados a ejecutar si NO se cumple la condición…
}
```

```
switch (variable){
    case(valor1):
        …enunciados a ejecutar para ese valor…
        break;
    case(valor2):
        …enunciados a ejecutar para ese valor…
        break;
    case(valor3):
        …enunciados a ejecutar para ese valor…
        break;
    default:
        …enunciados a ejecutar cuando el valor no
        coincida con ninguno de los anteriores…
        break;
}
```

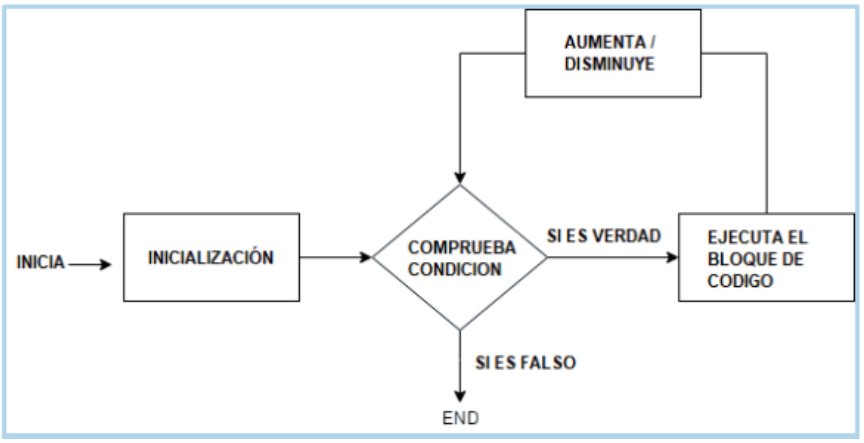
Operadores lógicos

Operador	Significado	Sintaxis
!	Negación	`if(!condición1) { …enunciados… } // los enunciados se ejecutan // si no se cumple la condición`
&&	Y	`if((condición1)&&(condición2)) { …enunciados… } // los enunciados se ejecutan // si se cumplen ambas condiciones`
||	O	`if((condición1)||(condición2)) { …enunciados… } // los enunciados se ejecutan // si se cumple una de las condiciones`

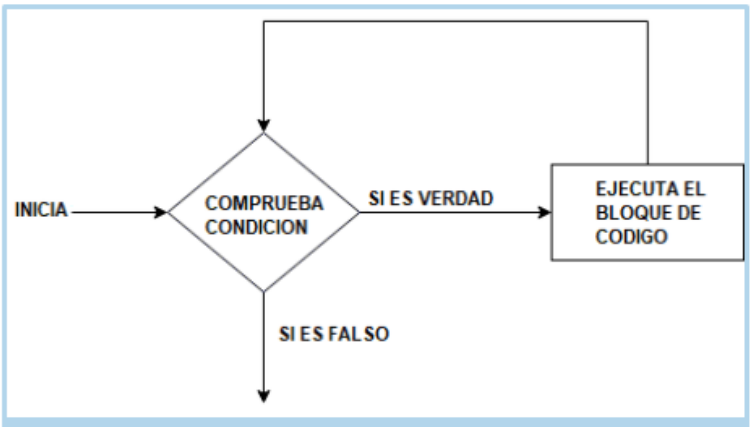
Bucles

```
for(int i = valor inicial; i <= valor final; i = i + paso)
{
        ....
        ....
    Bloque de Instrucciones....
        ....
        ....
}
```

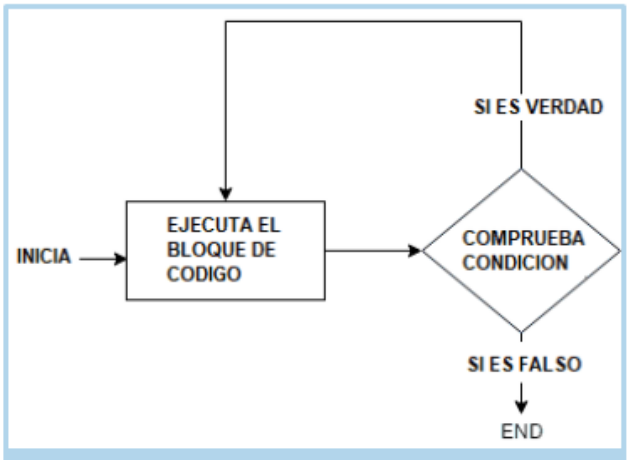
```
for (condición de inicialización, condición de prueba;
                              incremento / decremento)
{
    declaracion(es)
}
```




```
while (condición booleana)
{
   declaraciones del bucle ...
}
```




```
do
{
    //El código dentro del do se imprime incluso
    //si la condición es falsa
    System.out.println("Valor de x :" + x);
    x++;
}
while (x < 20);
```



EXTRA

```
do{
                System.out.print("Ingresa tu
nombre: ");
                nombre = input.nextLine();
                }while(nombre.equals(""));
        return nombre;
        }
```

```
do{
                System.out.print("Ingresa tu
promedio: ");
                promedio = input.nextInt();
                input.nextLine();
        }while(promedio < 0 || promedio > 10);
                return promedio;
}
```

```
if (usuarioGuardado.equals(username) &&
passwordGuardado.equals(password)) {
                        ingresoExitoso = true;

                } else {
        numeroIntentos++;
        if (numeroIntentos <= 2) {
        System.out.println("Usuario o clave
incorrectos, vuelva a intentar");
                        } else {

System.out.println("Usuario o clave
incorrectos");
                        }
                }
```

```
if (totalCorredores > 0) {
                        System.out.println("El
ganador es: " + numGanador + " y tardo: " +
tiempoGanador + " segundos");
                        float promedioEnSegundos =
totalTiempo / totalCorredores;
                        System.out.println("El
promedio de los corredores en segundos es: " +
promedioEnSegundos);
                        float porcentaje =
corredoresRapidos * 100 / totalCorredores;
                        System.out.println("El
porcentaje menos de una hora es " + porcentaje);
                } else {
System.out.println("Terminado");}
```

```
do {
System.out.println("Ingresa el curso:  ");
curso =input.next().charAt(0);
} while (curso != 'a' && curso != 'b' && curso
!= 'c' && curso != 'd');
```

```
if (viajesAdicionales <= TOPE) {
        return ABONO + (viajesAdicionales
*VIAJE_ADICIONAL);
                } else { int subtotal = ABONO +
(viajesAdicionales * VIAJE_ADICIONAL);
        return (int) (subtotal - (subtotal * 0.10)); }
```

```
public static int devolverSegundos(int
horas, int minutos, int segundos) {
                int total = horas * 3600 +
minutos * 60 + segundos;

                return total;
        }
```

```
do {
System.out.println("Bienvenido! ¿Desea calcular
el precio total del catering para su fiesta? (s/n)");
ingreso = input.nextLine().toLowerCase();
if (!ingreso.equals("s") && !ingreso.equals("n")) {

System.out.println("Ingreso incorrecto");
                        }} while (!ingreso.equals("s")
&& !ingreso.equals("n"));
```

```java
ArrayList<Integer> tabla = new ArrayList<Integer>();
for (int i = 0 ; i <= 10; i++) {
    tabla.add(0);
}
System.out.println(tabla);

for (Alumno alumno: alumnos) {
    int nota = alumno.getNota();
    tabla.set(nota, tabla.get(nota) + 1);
}

System.out.println(tabla);
```

```java
private static ArrayList<Alumno> mejoresAlumnos(ArrayList<Alumno> alu
    /*
     * Dado un AL de alumnos devolver otro AL con los alumnos que hay
     * la mejor nota (puede ser uno o varios).
     */
    ArrayList<Alumno> mejores = new ArrayList<Alumno>();
    int max = -1;
    for (Alumno alumno: alumnos) {
        if (alumno.getNota() >= max) {
            if (alumno.getNota() > max) {
                max = alumno.getNota();
                mejores.clear();
            }
            mejores.add(alumno);
        }
    }

    return mejores;
```

```java
ArrayList<Integer> frecuentes = new ArrayList<Integer>();
for (int i = 0 ; i <= 10 ; i++) {
    if (tabla.get(i) >= max) { // 5 y 9
        if (tabla.get(i) > max) { // 5
            max = tabla.get(i);
            frecuentes.clear();
        }
        frecuentes.add(i);
    }
```

```java
int max = -1;
ArrayList<Integer> frecuentes = new ArrayList<Integer>();
for (int i = 0 ; i <= 10 ; i++) {
    if (tabla.get(i) > max) { // 5
        max = tabla.get(i);
        frecuentes.clear();
        frecuentes.add(i);
    }
    else if (tabla.get(i) == max) { // 9
        frecuentes.add(i);
    }
```

```java
private static ArrayList<Alumno> mejoresAlumnos(ArrayList<Alumno> alumnos) {
    /*
     * Dado un AL de alumnos devolver otro AL con los alumnos que hayan obtenido
     * la mejor nota (puede ser uno o varios).
     */
    ArrayList<Alumno> mejores = new ArrayList<Alumno>();
    int max = -1;
    for (Alumno alumno: alumnos) {
        if (alumno.getNota() >= max) {
            if (alumno.getNota() > max) {
                max = alumno.getNota();
                mejores.clear();
            }
            mejores.add(alumno);
        }
    }

    return mejores;
}
```

THP-2021-1C_Clases de apoyo

https://github.com/elyelin/ort-ts1-tp2

Enunciado

Para un correcto seguimiento de la situación sanitaria actual, el ministerio de salud requiere un sistema para el control de la vacunación de la población.

El vacunatorio central tiene un listado de personas empadronadas para vacunarse y otro con los vacunatorios habilitados. También hay un listado donde quedan quienes hayan cumplido el esquema completo de vacunación (quienes ya tengan aplicadas las tres dosis de vacunas).

Cada empadronado cuenta con un dni (único) y una lista con las vacunas que se aplicó.

De cada vacuna conocemos su marca y el número de serie.

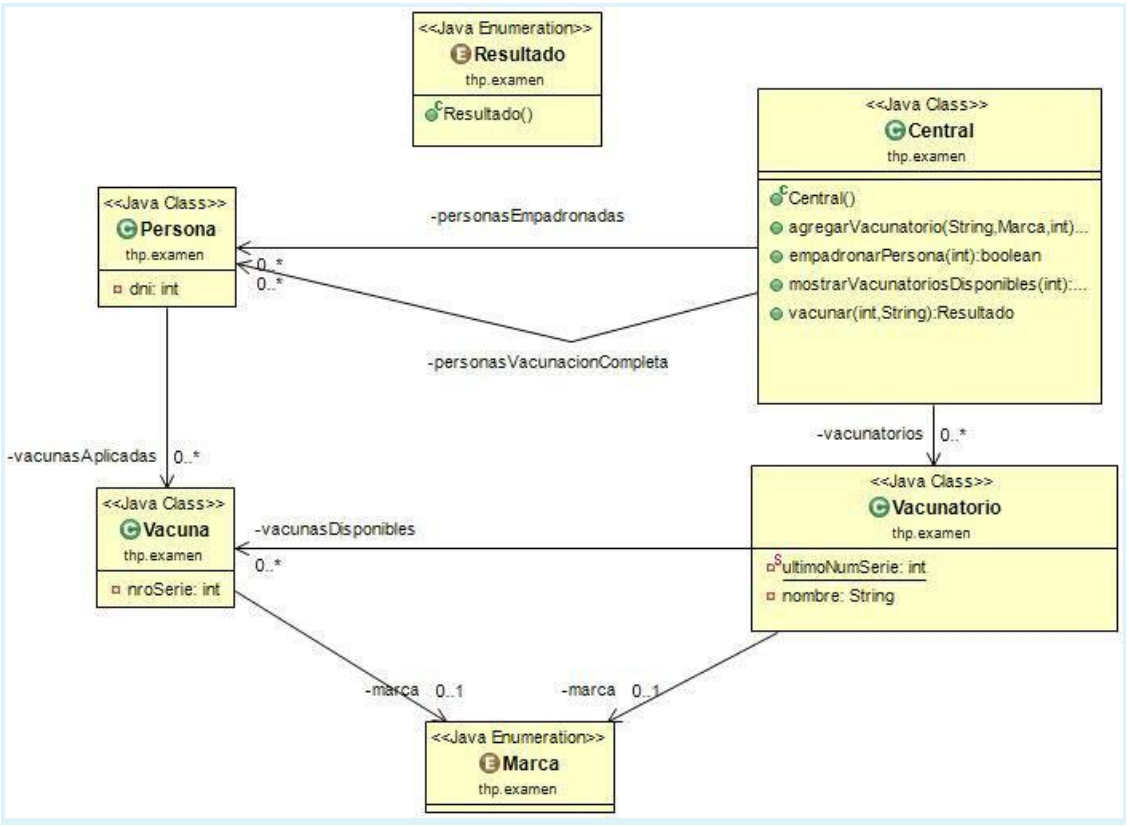
Cada vacunatorio aparte de su nombre, la marca de vacunas que usa y el último número de serie de vacunas que es un número correlativo, tiene una lista con las vacunas disponibles, las mismas tienen marca y número de serie . (Para esto tenes una ayuda en el enunciado)

Sabemos también que el registro de vacunas que tiene cada empadronado contiene las vacunas que le fueron aplicadas (con marca y número de serie)

Para confeccionar las funcionalidades del sistema se deben tener en cuenta los siguientes requisitos:

- Por el momento se aplican vacunas únicamente hasta la tercera dosis.
- Si una persona no tiene vacunas aplicadas, se puede aplicar cualquiera (es decir, cualquier marca que tenga dosis disponibles).
- Si una persona se ha aplicado previamente una sola dosis, la segunda debe ser de la misma marca que la primera.
- Si una persona se aplicó ya dos dosis, la tercera debe ser siempre diferente a la segunda.

El siguiente es el diagrama de clases con los atributos y los métodos a desarrollar, tene en cuenta que podés definir todos los métodos que consideres necesarios para mantener el correcto encapsulamiento y distribución de responsabilidades entre las clases del sistema.



1. empadronarPersona. Dado un dni especificado por parámetro. Debe agregar a la persona en el padrón, la misma no debe estar empadronada ni estar en la lista de personas con esquema completo. Devuelve un booleano que indica si pudo empadronar a la persona
2. agregarVacunatorio, dado un nombre de vacunatorio, una marca y una cantidad de vacunas, crear el vacunatorio en caso que el nombre no exista.
3. mostrarVacunatoriosDisponibles. Dado un dni especificado por parámetro, debe mostrar la lista de vacunatorios que tienen disponible la vacuna que le corresponde a la persona (si no existe el dni en el padrón mostrar DNI_INVALIDO).
4. vacunar. Dado un dni, el nombre de un vacunatorio , se debe procesar la vacunación de la persona. Si es posible vacunarla, registrar la aplicación de la vacuna y, si además completa el esquema, moverlo junto con los demás que completaron el esquema de vacunación. El método debe retornar:

- VACUNACION_COMPLETA, si la persona tiene ya las 3 dosis aplicadas.
- DNI_INVALIDO, si no existe el dni en el listado de personas empadronadas.
- VACUNATORIO_INEXISTENTE, si no existe el vacunatorio especificado.
- VACUNA_NO_DISPONIBLE, si no se encuentran vacunas compatibles para la persona en ese vacunatorio.
- VACUNACION_OK, si se puede vacunar. Recordá que se debe actualizar la lista de vacunas aplicadas de la persona y las dosis disponibles para la vacuna que se aplicó y mover a la persona a la lista de esquema completo en caso de ser necesario.

Para facilitarte la creación de vacunatorios y sus vacunas te dejamos el siguiente código para que lo incorpores en la clase que corresponda.

```
public Vacunatorio(String nombre, Marca marca, int cantidad) {
this.nombre = nombre;
this.marca = marca;
this.vacunasDisponibles = new ArrayList<>();
agregarLote(cantidad);
}
public void agregarLote(int cantidad) {
for (int i = 0; i < cantidad; i++) {
vacunasDisponibles.add(new Vacuna(marca, ultimoNumSerie++));
```

Test de la solución

La salida debe ser similar a la siguiente:

```
agregarVacunatorio(Rural, PFIZER, 5): true
agregarVacunatorio(Centro Islamico, MODERNA, 5): true
agregarVacunatorio(River, SPUTNIK, 5): true
agregarVacunatorio(River, SPUTNIK, 5): false

empadronarPersona(40111222): true
empadronarPersona(40111222): false
empadronarPersona(42333444): true
Vacunatorios disponibles para Persona [dni=40111222, vacunasAplicadas=[]]
```

Vacunatorio [nombre=Rural]

Vacunatorio [nombre=Centro Islamico]

Vacunatorio [nombre=River]

vacunar(40111222, Rural): VACUNACION_OK

Vacunatorios disponibles para Persona [dni=40111222, vacunasAplicadas=[Vacuna [marca=PFIZER, nroSerie=1]]]

Vacunatorio [nombre=Rural]

vacunar(40111222, Rural): VACUNACION_OK

Vacunatorios disponibles para Persona [dni=40111222, vacunasAplicadas=[Vacuna [marca=PFIZER, nroSerie=1], Vacuna [marca=PFIZER, nroSerie=2]]]

Vacunatorio [nombre=Centro Islamico]

Vacunatorio [nombre=River]

vacunar(40111222, Centro Islamico): VACUNACION_OK

vacunar(40111222, River): VACUNACION_COMPLETA

vacunar(99999999, River): DNI_INVALIDO

vacunar(42333444, Boca): VACUNATORIO_INEXISTENTE