

Practica Programacion Orientada a Objetos

Curso 2020-2021

Francisco Sanchez Jimenez

fransanjiz@hotmail.com

telefono: UK (07849251155)

Esta versión del programa es una versión para desarrolladores, tiene habilitadas funciones que no deben estar presentes en la versión final, en ésta el único método público de la clase clínica debería de ser iniciar() y a partir de aquí el usuario identificado puede acceder a los métodos propios del nivel de acceso correspondiente a su nivel jerárquico.

Los métodos en la clase clínica, cuando pueden cambiar el estado de otro objeto pide por parámetro el usuario, que tiene que corresponder con la acción que se intenta ejecutar, así, un administrador no podrá intentar realizar una prueba o analizarla y un técnico no podrá intentar eliminar o agregar un usuario.

La jerarquía de herencia se ha construido en tanto que se entiende que que todos los usuarios van a ser personas y se ha construido para ello una base de datos, individualizada en una clase aparte (BBDD) para mejorar la cohesión y disminuir el acoplamiento, compuesta de un HashMap que con un conjunto de clave-valor almacena al identificador único de cada usuario (su DNI) con el objeto persona propio de este identificador.

Todas las clases implementan la interfaz Serializable, de esta manera se puede convertir un objeto BBDD en un stream de números binarios para ser volcado en un archivo fuera del programa (o recuperado).

Esta función no está implementada en la versión actual del programa.

Tampoco se ha limitado el número que representa al DNI de forma alguna.

La base de datos mantiene un recuento de los usuarios con 3 variables primitivas int y es capaz de devolver una lista de pacientes, técnicos, enfermeros o administradores sacando los valores de clave de Hashmap y recorriéndolos uno a uno, creando una lista cuando la variable polimórfica que recorre esta última corresponde al tipo pedido en la función.

La base de datos también contiene una lista de los tratamientos que se han

efectuado en la clínica y una instancia de un objeto individual de clase Nevera, que abstrae la lógica detrás del almacenamiento de las vacunas y de la actualización del stock de este.

Estos métodos, aunque públicos, son inaccesibles en el programa fuera de la clase BBDD y estos son alcanzables solo en el menú administrador en la clase clínica, que, como indicado anteriormente necesita como parámetro una instancia de clase Administrador.

Los tratamientos que se pueden efectuar en la clínica pueden ser de dos tipos, puede un usuario someterse a una batería de pruebas o puede vacunarse. En mi opinión, creo que se da a entender en el enunciado de la práctica que estos tienen que ser árboles jerárquicos individuales, pero, desde un punto de vista más abstracto ambos comparten muchas similitudes y, por lo tanto los he anidado en una clase abstracta general llamada Tratamiento, que alberga los atributos comunes a todos ellos, como puede ser el paciente que se va a someter a dicho tratamiento (prueba o vacuna), la fecha en la que se realiza; no cuando se pide, dicho tratamiento, el enfermero y el técnico asignados. De esta manera se mantiene una relación entre los caracteres involucrados en la totalidad del proceso y facilita la búsqueda de información a través del programa.

Una vez más esta decisión ayuda a la hora de tratar errores en el código y explota el polimorfismo y las correcciones en tiempo de ejecución.

Quizás el nombre sea lo más inadecuado de la decisión ya que no se ajusta a la realidad el llamar a una vacuna "tratamiento", ya que no trata, si no que previene, pero a falta de un término mejor la clase cumple su función.

Las clases que heredan de esta clase abstracta son, Vacunas y Pruebas, ambas abstractas.

De Vacunas heredan:

-Pfizer y Moderna:

con una restricción dada por un valor estático en una clase abstracta de

apoyo llamado redundantemente "estático" donde se puede alterar todos estos parámetros, actualmente está restricción impone un total de 21 días pero si tuviese que ajustarse a la realidad (se ha reducido este tiempo de espera en la actualidad) solo tendría que cambiarse en un punto del código.

-JJ:

esta clase difiere en las anteriores en que con una dosis se consigue completa inmunización, para lograr adaptar esta capacidad se ha cambiado, modificado y sobre escrito los métodos que simbolizan la vacunación, haciendo que se reste la totalidad de las dos dosis del objeto vacuna JJ y que el estado del paciente pase a ser de completamente vacunado. Esto se comprueba también en tiempo de ejecución con la habilidad polimórfica heredada de la clase Vacunas, haciendo estas 3 clases completamente compatibles entre sí.

Tengo que recalcar que toda la lógica de vacunación es exclusiva del enfermero y que el "técnico asignado" se deja sin inicializar.

Tomando por otra parte la clase Pruebas, extiende Tratamiento agregando 3 atributos que controlan el estado en cada momento de la prueba a realizar. Estos tres atributos son del tipo primitivo boolean y se denominan resultado, procesado y ejecutado.

El flujo de trabajo se sucede, en este orden:

- 1º El administrador crea la prueba y la asigna (esto se verá más en profundidad cuando se analice la clase Clínica)
- 2º Cuando el enfermero empieza a trabajar, coge la prueba, comprueba a qué subclase corresponde y aplica los métodos correspondientes cambiado el estado "ejecutado" a la prueba.
- 3º Cuando el técnico empieza a trabajar comprueba qué pruebas en su lista semanal tienen el valor a true en "ejecutado", una vez encontrado un trabajo trabajable empieza a procesarlo, de nuevo

el polimorfismo determina el flujo de trabajo.

Si es una prueba detectora de virus y da positivo manda una alerta al módulo de confinamiento a través de la clínica asignada a su misma persona y marca acordemente los valores de resultado y procesado.

Para ambos trabajadores la lista semanal de trabajo se actualiza al comprobar si tiene espacio para otro trabajo, el método correspondiente devuelve un valor confirmativo si el tamaño de esta lista es menor al impuesto a la clase de trabajador, si no fuese menor se recorrería la lista buscando las pruebas con un tiempo superior a 7 días y eliminándolas de la lista planSemanal para almacenarlo en el de tratamientos. Dando lugar a otra prueba semanal a ser almacenada.

En el caso del enfermero también se comprueba si el tratamiento es una vacuna.

Como no hay restricciones al número de vacunas las acepta directamente.

Para ayudar a repartir la carga y que no se le asigne vacunas siempre al mismo enfermero hay una variable estática llamada umbral en la clase clínica.

Esta variable se inicializa a 1 en el momento de crear la clínica y una vez que se intenta asignar una vacuna se utiliza para recorrer el total del grupo de enfermeros, se compara con el número de vacunas asignada a cada uno y hace aceptar a aquel que tenga un número inferior a este umbral. Si después de una iteración completa esta vacuna todavía no se ha podido asignar crece en uno el valor de umbral y se recorre la lista una vez más.

Con respecto al caso de los análisis serológicos, aunque se pueden pedir individualmente en la clase clínica, lo más normal es que las peticiones salgan del módulo de confinamiento.

En la clase denominada ModuloConfinamiento se lleva a cabo todo el seguimiento de los pacientes que han dado positivo, como he comentado antes la alerta se

dispara desde el momento en que el resultado, procesado por el técnico, da positivo y a través de la clínica llega al módulo de confinamiento.

Estas alertas se mantienen en una lista hasta que el administrador decide procesarla, al procesarla hace ciertas comprobaciones, como por ejemplo que no haya sido una prueba repetida, recordemos que las pruebas de antígenos no tienen límite de días. Al aceptar la prueba la guarda en un objeto de clase privada denominado nodoConfinamiento. Este representa un día completo de la clínica, donde se almacenan las alertas procesadas y se almacena en una lista para su posterior consulta, por ejemplo, cuando se quiera hacer el seguimiento de los pacientes en confinamiento.

Por ello el nodo se compone de 3 listas LinkedList de tipo pruebas; procesar, no_contestado y completado.

Estas pruebas van cambiando de listas conforme se las va procesando y al finalizar el día el ModuloConfinamiento las recorre para devolverlas a su estado original.

Al día siguiente estarán listas para volver a recorrerse.

Al día siguiente se recorre la lista y se mueven los nodos superiores a 10 días de antigüedad a una lista llamada pendienteSerologico.

Esta se puede recorrer para asignar los análisis correspondientes, si se consigue asignar correctamente se elimina de la lista, en caso contrario permanece en ella.

En la documentación explícita de la clase he dado una visión más profunda de ésta en la que hago unas observaciones sobre la capacidad de ampliación de este módulo, aun así, creo que cumple con lo que se pide de él en la práctica.

Por otra parte, la clase Clínica representa a la clínica en sí, desde ésta se administran el resto de los módulos y opciones.

Esta clase se compone de una instancia de los objetos base de datos (BBDD), una instancia del ModuloConfinamiento y una variable de referencia Person que representa el usuario "logueado" en un momento dado en la clínica.

También almacena una variable umbral dinámica, de implementarse la funcionalidad de carga de base de datos desde un archivo ésta también tendría que actualizarse con el último valor para ahorrar el coste de cálculo de éste. Si no, al intentar asignar vacunas éste mismo valor se irá auto actualizándose pues se inicia en el valor más pequeño y aumentaría hasta encontrar a un enfermero con menos vacunas pendientes que este valor.

Con respecto a sus métodos y funciones el único método que debería de permanecer public es el de iniciar(). En este se comprueba el estado de la variable baseDatos. De ser null se pide si se quiere cargar la base de pruebas (correspondiente al modo debug) o si se quiere iniciar una nueva. En este punto se podría implementar la carga desde archivo.

Al crear la base de datos se procede con el método cargaUsuario(), este comprueba que haya usuarios en la base de datos y de no haberlos comienza creando un usuario Administrador y pidiendo los datos correspondientes. Una vez finalizado esto se carga el usuario Administrador como usuario de la clínica y se carga el menú correspondiente a Administrador.

Si hubiese habido usuarios en la base de datos pide al controlador del programa que introduzca el DNI (el propio se deduce) y carga este usuario en la clínica.

En este punto sería fácil comprobar que el usuario es el correcto pidiendo una contraseña que puede estar almacenada en la propia ficha del usuario, aunque para una práctica no parece merecer el tiempo extra de comprobaciones.

Una vez más, se comprueba en tiempo de ejecución el tipo de usuario y se carga acordemente.

Procediendo con los menús;

Menú administrador:

1 Gestión de usuarios

Permite la creación y eliminación de usuarios.

2 Asignación de tratamientos

Pruebas y vacunas

3 Visualización de datos

Una visión general del estado de la clínica o de un usuario en específico.

4 Modulo Confinamiento

Permite el acceso a las funciones respectivas del módulo de confinamiento

5 Actualización del stock de vacunas

Accede a la "nevera" a través de la base de datos y actualiza el stock acordemente

6 Visualización planificación de vacunas

Recorre la lista de prioridades e imprime los datos

7 Cerrar sesión

Pone a null el valor de usuario de la clínica y termina la sesión.

Menú trabajador

1 Visualización de datos plan Semanal

Imprime los datos del plan Semanal

2 Visualización de tratamientos completados

Imprime los datos de los tratamientos completados

3 Visualización datos de paciente

Pide un DNI de un paciente y muestra la información de éste

4 Registro y actualización de tratamientos

Comprueba la instancia de usuario en tiempo de ejecución;

de ser técnico comienza a procesar las pruebas

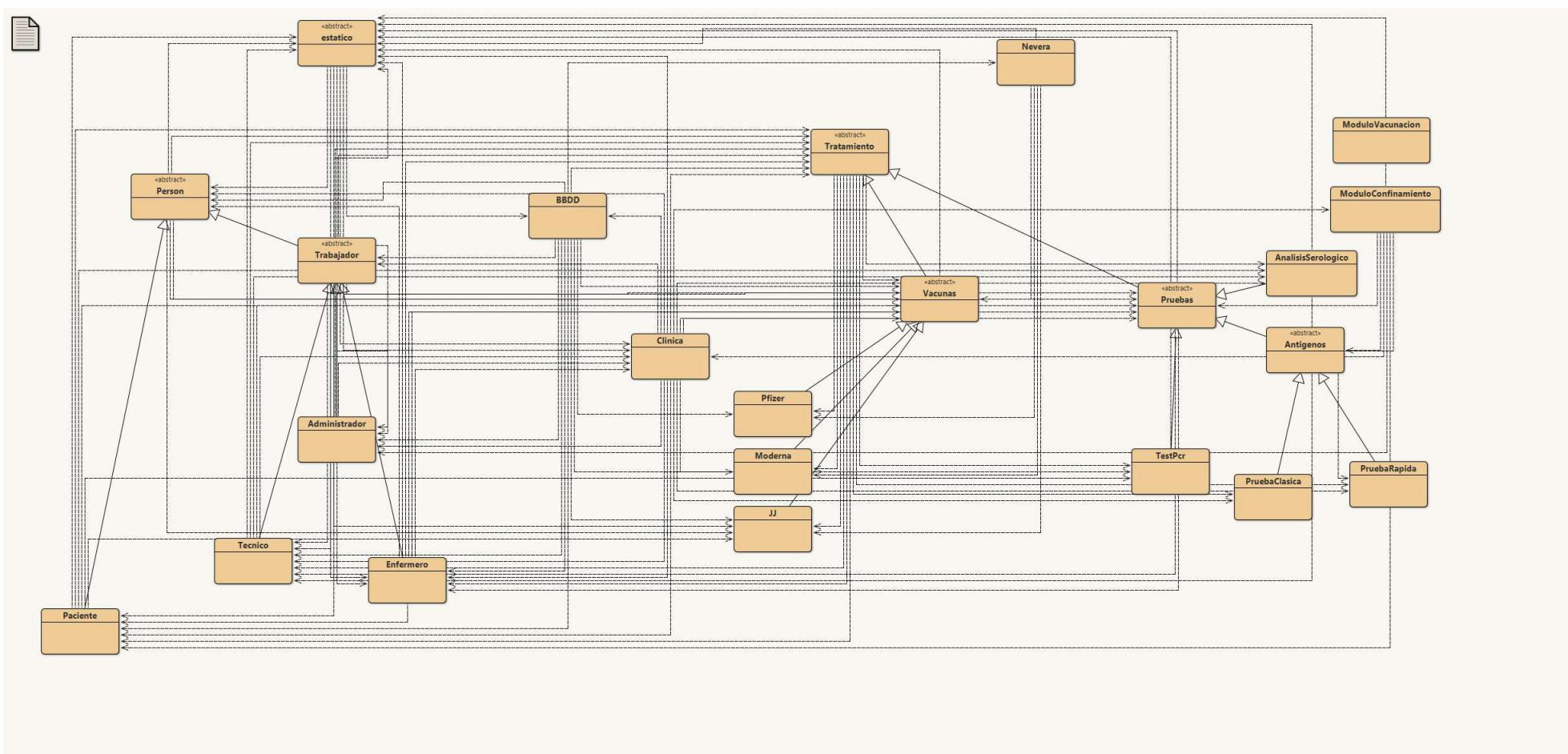
de ser enfermero pregunta si quiere trabajar en las

vacunas o en las pruebas y actualiza su trabajo

acordemente.

Menú Paciente

A un paciente la clínica no le permite ver sus datos, le pide que contacte con
un trabajador.[▲](#)



Package <Unnamed>

Class Summary

Class	Description
Administrador	Esta clase representa al administrador de la clinica, las funciones del administrador estan declaradas en la clinica y esta comprueba que el usuario es un administrador para permitir la manipulacion de esta.
AnalisisSerologico	Analisis serologico hereda de prueba aunque es un poco mas especial que el resto, esta prueba no determina si se tiene el virus, si no si se hayan presentes en el organismo los anticuerpos para este.
Antigenos	Abstract class Antigenos - clase abstracta, representa el comun de las pruebas de antigenos, las cuales se pueden hacer diariamente.
BBDD	La base de datos de la que se nutre la clase clinica y donde se almacena toda la informacion, implementa la interfaz Serializable para poder hacer una copia y mandarlo como stream de digitos binarios a un archivo fuera del entorno de bluej y a la inversa, para cargar un objeto base de datos de un archivo independiente y restaurar la informacion de una ejecucion anterior.
Clinica	Clase principal del programa, desde aqui se administra y se consultan los estados de este.
Enfermero	
estatico	Clase que almacena funciones, metodos y parametros de utilidad general en el programa.
JJ	Write a description of class JJ here.
Moderna	Write a description of class Moderna here.
ModuloConfinamiento	Esta clase se encarga de los confinamientos, utiliza una clase privada NodoConfinamiento, compuesta de 3 listas las cuales van pasandose las pruebas segun se confirme que el paciente esta aislando.
ModuloVacunacion	Write a description of class ModuloVacunacion here.
Nevera	Clase que representa una nevera donde se almacenan las vacunas
Paciente	Paciente es una subclase de Person, se construye con un digito dni que representa un valor unico, consta de dos atributos extras con respecto a la clase persona, primeraDosis que representa la situacion de que el paciente haya recibido una dosis de dos y vacunado, que representa la totalidad de la vacuacion.
Person	Abstract class Person - Superclase, define los atributos de un objeto abstracto persona Atributos principales designados en la clase: Identificador Nombre Apellido Edad lista de tratamientos (el significado de esta lista varia en funcion de la subclase)

Pfizer	Write a description of class Pfizer here.
PruebaClasica	Esta clase solo diferencia entre tipos de pruebas de antigenos y no contiene ninguna diferencia con respecto a la clase padre
PruebaRapida	Esta clase solo diferencia entre tipos de pruebas de antigenos y no contiene ninguna diferencia con respecto a la clase padre
Pruebas	Abstract class Pruebas - clase de la que derivan las pruebas, esta a su vez hereda de tratamientos consta de 3 atributos mas con respecto a Tratamientos; resultado, procesado y ejecutado.
Tecnico	Funciones del tecnico: -Visualizacion de datos de los pacientes asignados -Registro y actualizacion de pruebas diagnosticas y vacunacion
TestPcr	Esta clase representa a un test PCR, impone un limite en los dias entre pruebas pero no tiene nada mas especial con respecto a otra prueba cualquiera de analisis de presencia viral
Trabajador	Abstract class Trabajador - write a description of the class here
Tratamiento	Abstract class Tratamiento - Esta clase representa de forma general cualquier procedimiento por el cual un paciente o usuario haya acudido a la clinica.
Vacunas	Abstract class Vacunas - Define unos patrones genericos para las 3 vacunas y los metodos aplicables como interfaz.

[PACKAGE](#) [CLASS](#) [TREE](#) [INDEX](#) [HELP](#)

[ALL CLASSES](#)

Class Administrador

```
java.lang.Object
Person
    Trabajador
        Administrador
```

All Implemented Interfaces:

Serializable

```
public class Administrador
extends Trabajador
```

Esta clase representa al administrador de la clinica, las funciones del administrador estan declaradas en la clinica y esta comprueba que el usuario es un administrador para permitir la manipulacion de esta.

Version:

(0.1) Administrador; funciones: -Gestion de usuarios: altas, bajas y modificaciones de todas las personas -Asignacion de pruebas diagnosticas a enfermeros y tecnicos y asignacion de vacunaciones a enfermeros -Visualizacion de datos de todas las personas registradas en el sistema -Visualizacion de pacientes asignados a cada enfermero/tecnico para pruebas diagnosticas y vacunaciones -Gestion de la programacion de pruebas serologicas tras los confinamientos. - Actualizacion del stock de vacunas. -Visualizacion de la planificacion tentativa de vacunas, a partir de los pacientes registrados en un momento determinado

Author:

(Francisco Sanchez)

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor	Description
<code>Administrador(int DNI, Clinica c)</code>	
<code>Administrador(Clinica c, int DNI, String nombre, String apellido1, String apellido2, int edad)</code>	

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type	Method	Description
void	<code>aceptar_Tratamiento(Tratamiento x)</code>	metodos interfaz para poder usar polimorfismo, sobreescritos en clases hijas
void	<code>add_Tratamiento(Tratamiento x)</code>	mutador, agrega un tratamiento a la lista de tratamientos
boolean	<code>planSemanal_espacioLibre()</code>	
boolean	<code>planSemanal_espacioLibre(int x)</code>	devuelve la comprobacion de si el objeto trabajador tiene espacio en su lista de trabajos semanales.

Methods inherited from class Trabajador

`agregar_planSemanal, alerta_Clinica, coger_prueba, imprimir_completados, imprimir_pacientes_Completados, imprimir_pacientes_planSemanal, imprimir_planSemanal, in_planSemanal, iterador_planSemanal, num_casos_PS, tiene_Trabajo`

Methods inherited from class Person

`contains_Tratamiento, formularioNuevoRegistro, get_apellido1, get_apellido2, get_edad, get_identificadorDNI, get_nombre, get_tratamientos, imprimir_Tratamientos, nueva_listaTratamientos, numero_tratamientos, poll_Tratamiento, remove_Tratamiento, set_apellido1, set_apellido2, set_edad, set_nombre, toString`

Methods inherited from class java.lang.Object

`clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

Administrador

```
public Administrador(int DNI,
Clinica c)
```

Administrador

```
public Administrador(Clinica c,
    int DNI,
    String nombre,
    String apellido1,
    String apellido2,
    int edad)
```

Method Detail

aceptar_Tratamiento

```
public void aceptar_Tratamiento(Tratamiento x)
```

Description copied from class: Trabajador

metodos interfaz para poder usar polimorfismo, sobreescritos en clases hijas

Specified by:

`aceptar_Tratamiento` in class `Trabajador`

planSemanal_espacioLibre

```
public boolean planSemanal_espacioLibre()
```

Specified by:

`planSemanal_espacioLibre` in class `Trabajador`

planSemanal_espacioLibre

```
public boolean planSemanal_espacioLibre(int x)
```

Description copied from class: Trabajador

devuelve la comprobacion de si el objeto trabajador tiene espacio en su lista de trabajos semanal. Comprueba si alguna creada, si no la crea. Comprueba si la carga semanal es menor que el limite y devuelve que hay espacio libre en caso afirmativo. En el caso de que siga sin haber devuelto un valor true recorre la lista y comprueba los dias que han pasado entre pruebas, si hay alguna mayor de 7 dias de antiguedad taponando la lista, la elimina y la agrega a la lista de tratamientos completados.

Overrides:

`planSemanal_espacioLibre` in class `Trabajador`

Parameters:

`x` - de tratamientos por semana

Returns:

puede aceptar otro tratamiento.

add_Tratamiento

```
public void add_Tratamiento(Tratamiento x)
```

Description copied from class: Person

mutador, agrega un tratamiento a la lista de tratamientos

Overrides:

`add_Tratamiento` in class `Person`

Parameters:

`x` - tratamiento

PACKAGE **CLASS** TREE INDEX HELP

ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class AnalisisSerologico

```
java.lang.Object
    Tratamiento
        Pruebas
            AnalisisSerologico
```

All Implemented Interfaces:

Serializable

```
public class AnalisisSerologico
extends Pruebas
```

Analisis serologico hereda de prueba aunque es un poco mas especial que el resto, esta prueba no determina si se tiene el virus, si no si se hayan presentes en el organismo los anticuerpos para este. Por lo tanto el resultado determina en este caso si se esta "inmunizado". Hay que prestar especial atencion para no mezclarlo con los otros.

Version:

(0.1)

Author:

(Francisco Sanchez)

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor	Description
<code>AnalisisSerologico()</code>	constructor del analisis serologico. el constructor llama a super() automaticamente.

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type	Method	Description
void	<code>analizar_prueba(Tecnico tecnico)</code>	metodo que sobreescribe al metodo de la clase padre analizar_prueba.

```
int      get_lapsoDias()
```

metodo accesror del tiempo en dias entre pruebas de tipo serologico.

Methods inherited from class Pruebas

```
get_ejecutado, get_procesado, get_resultado, hacer_prueba, set_resultado
```

Methods inherited from class Tratamiento

```
aceptado, cuota_tiempo, get_dni_Paciente, get_Enfermero, get_fecha, get_Paciente,
get_Tecnico, get_tipoTratamiento, remove_Enfermero, remove_Paciente,
remove_Tecnico, remove_Tratamiento, set_enfermeroAsignado, set_paciente,
set_tecnico, timeStamp
```

Methods inherited from class java.lang.Object

```
clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Constructor Detail

AnalisisSerologico

```
public AnalisisSerologico()
```

constructor del analisis serologico. el constructor llama a super() automaticamente.

Method Detail

analizar_prueba

```
public void analizar_prueba(Tecnico tecnico)
```

metodo que sobreescribe al metodo de la clase padre analizar_prueba. La clase difiere del resto y por lo tanto necesita un trato especial.

Overrides:

analizar_prueba in class Pruebas

Parameters:

el - tecnico que va a analizar la prueba

get_lapsoDias

```
public int get_lapsoDias()
```

metodo accesor del tiempo en dias entre pruebas de tipo serologico.

Specified by:

get_lapsoDias in class Pruebas

Returns:

estatico.SEROLOGICO

PACKAGE **CLASS** TREE INDEX HELP

ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class Antigenos

```
java.lang.Object
    Tratamiento
        Pruebas
            Antigenos
```

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

PruebaClasica, PruebaRapida

```
public abstract class Antigenos
extends Pruebas
```

Abstract class Antigenos - clase abstracta, representa el comun de las pruebas de antigenos, las cuales se pueden hacer diariamente.

Version:

(0.1)

Author:

(Francisco Sanchez)

See Also:

Serialized Form

Field Summary

Fields

Modifier and Type	Field	Description
(package private) int	lapsoDias	

Constructor Summary

Constructors

Constructor	Description
Antigenos ()	

Method Summary

All Methods	Instance Methods	Concrete Methods
Modifier and Type	Method	Description
int	<code>get_lapsoDias()</code>	implementa el metodo abstracto de la clase padre.

Methods inherited from class Pruebas

`analizar_prueba, get_ejecutado, get_procesado, get_resultado, hacer_prueba, set_resultado`

Methods inherited from class Tratamiento

`aceptado, cuota_tiempo, get_dni_Paciente, get_Enfermero, get_fecha, get_Paciente, get_Tecnico, get_tipoTratamiento, remove_Enfermero, remove_Paciente, remove_Tecnico, remove_Tratamiento, set_enfermeroAsignado, set_paciente, set_tecnico, timeStamp`

Methods inherited from class java.lang.Object

`clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

lapsoDias

`final int lapsoDias`

See Also:

[Constant Field Values](#)

Constructor Detail

Antigenos

`public Antigenos()`

Method Detail

get_lapsoDias

```
public int get_lapsoDias()
```

implementa el metodo abstracto de la clase padre.

Specified by:

`get_lapsoDias` in class `Pruebas`

Returns:

`lapsoDias`

[PACKAGE](#) [CLASS](#) [TREE](#) [INDEX](#) [HELP](#)

[ALL CLASSES](#)

SUMMARY: NESTED | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Class BBDD

java.lang.Object
BBDD

All Implemented Interfaces:

Serializable

```
public class BBDD
extends Object
implements Serializable
```

La base de datos de la que se nutre la clase clinica y donde se almacena toda la informacion, implementa la interfaz Serializable para poder hacer una copia y mandarlo como stream de digitos binarios a un archivo fuera del entorno de bluej y a la inversa, para cargar un objeto base de datos de un archivo independiente y restaurar la informacion de una ejecucion anterior. No implementada esta ultima funcion en la version 0.1. Utiliza una estructura HashMap que almacena una tupla de valores para guardar un registro de los usuarios de la clinica, . Utiliza una instancia de la clase Nevera, que guarda la informacion del stock de vacunas, como su nombre indica, representa la idea de una nevera, donde se guardan los distintos tipos de vacunas. Tambien hace uso de 4 variables int que guardan un contador de los tipos de usuarios registrados en la base de datos.

Version:

(0.1)

Author:

(Francisco Sanchez)

See Also:

Serialized Form

Field Summary

Fields

Modifier and Type	Field	Description
(package private) int	numeroAdministradores	
(package private) int	numeroEnfermeros	
(package private) int	numeroPacientes	
(package private) int	numeroTecnicos	

Constructor Summary

Constructors	
Constructor	Description
<code>BBDD ()</code>	Constructor for objects of class BBDD

Method Summary

All Methods	Instance Methods	Concrete Methods
Modifier and Type	Method	Description
void	<code>add_registro(int dni, Person datosPersona)</code>	
void	<code>add_Tratamiento(Tratamiento trat)</code>	
void	<code>agrega_JJ(int x)</code>	
void	<code>agrega_Moderna(int x)</code>	
void	<code>agrega_Pfizer(int x)</code>	
boolean	<code>contains_registro(int dni)</code>	
void	<code>eliminar_registro(int dni)</code>	
<code>HashMap</code>	<code>get_archivo()</code>	
<code>Nevera</code>	<code>get_Nevera()</code>	
int	<code>get_numeroJJ()</code>	
int	<code>get_numeroModerna()</code>	
int	<code>get_numeroPfizer()</code>	
<code>Person</code>	<code>get_registro(int dni)</code>	
<code>Vacunas</code>	<code>get_vacuna_Random()</code>	saca una vacuna aleatoria de la nevera
<code>JJ</code>	<code>get_vacunaJJ()</code>	
<code>Moderna</code>	<code>get_vacunaModerna()</code>	
<code>Pfizer</code>	<code>get_vacunaPfizer()</code>	
<code>LinkedList<Vacunas></code>	<code>get_vacunas()</code>	
<code>LinkedList<Trabajador></code>	<code>lista_Enfermero()</code>	Devuelve uan lista con los enfermeros extraidos de la lista original
<code>LinkedList<Paciente></code>	<code>lista_Pacientes()</code>	Devuelve una lista con los pacientes extraidos de la lista original
<code>LinkedList<Trabajador></code>	<code>lista_Tecnicos()</code>	Devuelve una lista con los

tecnicos extraidos de la lista
original

int	numeroAdministradores()	
int	numeroEnfermeros()	
int	numeroPacientes()	
int	numeroTecnicos()	
int	numeroVacunas()	metodos accesores del objeto Nevera
int	size_registro()	

Methods inherited from class java.lang.Object

clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

numeroTecnicos

int numeroTecnicos

numeroEnfermeros

int numeroEnfermeros

numeroPacientes

int numeroPacientes

numeroAdministradores

int numeroAdministradores

Constructor Detail

BBDD

```
public BBDD()
```

Constructor for objects of class BBDD

Method Detail

add_registro

```
public void add_registro(int dni, Person datosPersona)
```

numeroTecnicos

```
public int numeroTecnicos()
```

numeroEnfermeros

```
public int numeroEnfermeros()
```

numeroPacientes

```
public int numeroPacientes()
```

numeroAdministradores

```
public int numeroAdministradores()
```

get_registro

```
public Person get_registro(int dni)
```

get_archivo

```
public HashMap get_archivo()
```

contains_registro

```
public boolean contains_registro(int dni)
```

eliminar_registro

```
public void eliminar_registro(int dni)
```

size_registro

```
public int size_registro()
```

add_Tratamiento

```
public void add_Tratamiento(Tratamiento trat)
```

get_vacunas

```
public LinkedList <Vacunas > get_vacunas()
```

numeroVacunas

```
public int numeroVacunas()
```

metodos accesores del objeto Nevera

get_vacunaJJ

```
public    JJ  get_vacunaJJ()
```

get_vacunaModerna

```
public    Moderna  get_vacunaModerna()
```

get_vacunaPfizer

```
public    Pfizer   get_vacunaPfizer()
```

agrega_JJ

```
public void agrega_JJ(int x)
```

agrega_Pfizer

```
public void agrega_Pfizer(int x)
```

agrega_Moderna

```
public void agrega_Moderna(int x)
```

get_Nevera

```
public    Nevera  get_Nevera()
```

get_numeroPfizer

```
public int get_numeroPfizer()
```

get_numeroModerna

```
public int get_numeroModerna()
```

get_numeroJJ

```
public int get_numeroJJ()
```

lista_Pacientes

```
public LinkedList <Paciente> lista_Pacientes()
```

Devuelve una lista con los pacientes extraidos de la lista original

lista_Tecnicos

```
public LinkedList <Trabajador> lista_Tecnicos()
```

Devuelve una lista con los tecnicos extraidos de la lista original

lista_Enfermero

```
public LinkedList <Trabajador> lista_Enfermero()
```

Devuelve uan lista con los enfermeros extraidos de la lista original

get_vacuna_Random

```
public Vacunas get_vacuna_Random()
```

saca una vacuna aleatoria de la nevera

Returns:

Vacunas

Class Clinica

java.lang.Object
Clinica

```
public class Clinica
extends Object
```

Clase principal del programa, desde aqui se administra y se consultan los estados de este.

Version:

(0.1)

Author:

(Francisco)

Field Summary

Fields

Modifier and Type	Field	Description
(package private) int	umbral	

Constructor Summary

Constructors

Constructor	Description
Clinica ()	Constructor for objects of class Clinica

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method	Description
void	actualizar_stockVacunas(Administrador t)	Metodo que actualiza el stock de vacunas, necesario que el usuario de la clinica en el momento sea un administrador (medida

		de seguridad y bloqueo)
void	alerta(Tecnico x, Pruebas y)	Manda la prueba con resultado positivo al modulo de confinamiento.
void	alertaVacuna(Trabajador x, Tratamiento y, boolean z)	Manda una senhal al modulo de vacunacion, si el parametro booleano es igual a true la senhal es de eliminacion de informacion, si la senhal es false manda una senhal al modulo para indicarle que el usuario ha sido ya vacunado y sacarlo de las colas de prioridad correspondientes.
void	altaUsuario(Administrador t, int DNI)	Metodo que crea usuarios por eleccion
void	asignar_prueba(int tipo, Paciente x, Administrador y)	metodo que asigna pruebas a los diferentes diferentes usuarios,
void	asignarTratamiento(Administrador t)	Metodo decorador del metodo asignar_prueba(), hace las veces de comprobador de estado de la clinica y configurador de parametros, tiene algunos metodos bloqueados a la espera de ser implementados (version 0.1)
void	asignarTratamiento(Administrador t, int DNIPaciente, int tipo)	metodo que asigna un tratamiento de forma directa, sin pedir parametros al usuario.
void	imprimir_datosUsuarioUnitario()	metodo que imprime los datos de un usuario concreto, lo implementa el menu de trabajador y el de administrador, comprueba tipo de objeto en tiempo de ejecucion (tipo de persona) e imprime la informacion especializada de la clase
void	iniciar()	Metodo principal de la clase, todas las funciones se controlan desde aqui
void	menuTrabajador(Trabajador x)	Menu de trabajadores, necesita que el usuario de la clinica sea un trabajador, comprueba el tipo de instancia en tiempo de ejecucion y aplica los metodos y las funciones propias de la clase final.
boolean	trabajadoresLibres()	metodo que comprueba que haya trabajadores con espacio en su plan semanal

Methods inherited from class java.lang.Object

`clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

umbral

```
int umbral
```

Constructor Detail

Clinica

```
public Clinica()
```

Constructor for objects of class Clinica

Method Detail

iniciar

```
public void iniciar()
```

Metodo principal de la clase, todas las funciones se controlan desde aqui

altaUsuario

```
public void altaUsuario(Administrador t, int DNI)
```

Metodo que crea usuarios por eleccion

asignar_prueba

```
public void asignar_prueba(int tipo, Paciente x, Administrador y)
```

metodo que asigna pruebas a los diferentes diferentes usuarios,

Parameters:

tipo - tipo de prueba, enumerada: 1 Serologico 2 PCR 4 Vacuna y 3 Antigeno

paciente - usuario al que asignar la prueba

usuario - de la clinica, medida de seguridad impuesta en tiempo de ejecucion, si la clase dada por parametro no es un admin e intenta acceder falla el programa.

asignarTratamiento

```
public void asignarTratamiento(Administrador t)
```

Metodo decorador del metodo asignar_prueba(), hace las veces de comprobador de estado de la clinica y configurador de parametros, tiene algunos metodos bloqueados a la espera de ser implementados (version 0.1)

trabajadoresLibres

```
public boolean trabajadoresLibres()
```

metodo que comprueba que haya trabajadores con espacio en su plan semanal

Returns:

boolean

asignarTratamiento

```
public void asignarTratamiento(Administrador t, int DNIPaciente, int tipo)
```

metodo que asigna un tratamiento de forma directa, sin pedir parametros al usuario. Implementado como una forma de acceder a las clases para el modo debug y pruebas. Este metodo hace un bypass a los comrpobadores de estados presentes en la cadena de sucesion de metodos presente en el programa principal No necesario en el programa final.

Parameters:

Administrador - , restringe el uso del metodo a los administradores

DNIPaciente - , el identificador del paciente que se va a someter a las pruebas

tipo - de tratamiento a crear y asignar.

actualizar_stockVacunas

```
public void actualizar_stockVacunas(Administrador t)
```

Metodo que actualiza el stock de vacunas, necesario que el usuario de la clinica en el momento sea un administrador (medida de seguridad y bloqueo)

Parameters:

administrador - de la clinica

menuTrabajador

```
public void menuTrabajador(Trabajador x)
```

Menu de trabajadores, necesita que el usuario de la clinica sea un trabajador, comprueba el tipo de instancia en tiempo de ejecucion y aplica los metodos y las funciones propias de la clase final. Permite que los enfermeros y tecnicos vean la carga de trabajo que tienen y que actualicen la informacion acordemente.

Parameters:

operario - de la clinica

imprimir_datosUsuarioUnitario

```
public void imprimir_datosUsuarioUnitario()
```

metodo que imprime los datos de un usuario concreto, lo implementa el menu de trabajador y el de administrador, comprueba tipo de objeto en tiempo de ejecucion (tipo de persona) e imprime la informacion especializada de la clase

alerta

```
public void alerta(Tecnico x, Pruebas y)
```

Manda la prueba con resultado positivo al modulo de confinamiento. los parametros sirven para restringir el flujo de informacion y lanzar fallo en tiempo de ejecucion si un usuario que no es de la clase adecuada intenta usar el metodo. Este metodo solo esta implementado en la clase Tecnico.

alertaVacuna

```
public void alertaVacuna(Trabajador x, Tratamiento y, boolean z)
```

Manda una senhal al modulo de vacunacion, si el parametro booleano es igual a true la senhal es de eliminacion de informacion, si la senhal es false manda una senhal al modulo para indicarle que el usuario ha sido ya vacunado y sacarlo de las colas de prioridad correspondientes. No implementado en la version 0.1

Class Enfermero

```
java.lang.Object
Person
    Trabajador
        Enfermero
```

All Implemented Interfaces:

`Serializable`

```
public class Enfermero
extends Trabajador
```

Version:

(0.1) Funciones del enfermero: -Visualizacion de datos de los pacientes asignados -Registro y actualizacion de pruebas diagnosticas y vacunacion

Author:

(Francisco Sanchez)

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor	Description
<code>Enfermero(int DNI, Clinica clinica)</code>	Constructor de la clase enfermero, requiere el identificador unico de la clinica como lugar de trabajo
<code>Enfermero(Clinica clinica, int DNI, String nombre, String apellido1, String apellido2, int edad)</code>	Constructor sobrecargado para crear un enfermero con todos los datos suministrado por parametro, sin uso en el programa final pero util para las opciones de debug y pruebas.

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type	Method	Description
<code>void</code>	<code>aceptar_Tratamiento(Tratamiento x)</code>	

`void add_Tratamiento(Tratamiento x)`

metodo sobreescrito de la clase padre,
hace comprobaciones extra y redirige el
tratamiento a la lista adecuada

`boolean planSemanal_espacioLibre()`

Metodo sobrecargado de la clase padre
Trabajador.

`boolean planSemanal_espacioLibre(int x)`

metodo sobreescrito de la funcion padre,
aplica las restricciones pertinentes a la
clase enfermero, valor almacenado en
clase estatico

`void realizar_Prueba(Pruebas pruebaApaciente)`

metodo sobreescrito, aplica una
restriccion diferente a los limites de
trabajo

`void remove_Tratamiento(Tratamiento x)`

Metodo que formaliza la accion de
realizar la prueba, cambia el estado de la
prueba a ejecutado, podria haber sido
sobrecargado para aceptar polimorfismo
con respecto al parametro (al trabajador)
pero por semantica me ha parecido que
era mejor mantener los metodos
separados, el de realizar la prueba en si y
el de analizarla.

`void trabajarPrueba()`

metodo sobrecargado de la clase padre
persona.

`void trabajarVacuna()`

metodo que hace trabajar al enfermero en
su lista de trabajos semanal

`int vacunacionesRestantes()`

Trabaja la lista de trabajos de vacunacion
por completo, comprueba la posiblidad de
vacunar antes de proceder

`void vacunar(Vacunas z)`

Devuelve el tamaño de la lista de
vacunaciones

Implementacion de la accion de vacunar a
un paciente, necesita como parametro
una vacuna que no este vacia, en base a
los valores de esta procede a eliminarla de
la lista de vacunas pendientes.

Methods inherited from class Trabajador

`agregar_planSemanal, alerta_Clinica, coger_prueba, imprimir_completados,`
`imprimir_pacientes_Completados, imprimir_pacientes_planSemanal,`
`imprimir_planSemanal, in_planSemanal, iterador_planSemanal, num_casos_PS,`
`tiene_Trabajo`

Methods inherited from class Person

```
contains_Tratamiento, formularioNuevoRegistro, get_apellido1, get_apellido2,
get_edad, get_identificadorDNI, get_nombre, get_tratamientos,
imprimir_Tratamientos, nueva_listaTratamientos, numero_tratamientos,
poll_Tratamiento, set_apellido1, set_apellido2, set_edad, set_nombre, toString
```

Methods inherited from class java.lang.Object

```
clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait
```

Constructor Detail

Enfermero

```
public Enfermero(int DNI,
                 Clinica clinica)
```

Constructor de la clase enfermero, requiere el identificador unico el la clinica como lugar de trabajo

Parameters:

DNI - identificador

clinica - lugar de trabajo

Enfermero

```
public Enfermero(Clinica clinica,
                  int DNI,
                  String nombre,
                  String apellido1,
                  String apellido2,
                  int edad)
```

Constructor sobrecargado para crear un enfermero con todos los datos suministrado por parametro, sin uso en el programa final pero util para las opciones de debug y pruebas.

Parameters:

clinica - lugar de trabajo

DNI - identificador unico

nombre - nombre del enfermero

apellido1 - apellido del enfermero

apellido2 - segundo apellido del enfermero

edad - edad del enfermero

Method Detail

vacunacionesRestantes

```
public int vacunacionesRestantes()
```

Devuelve el tamaño de la lista de vacunaciones

Returns:

size

vacunar

```
public void vacunar(Vacunas z)
```

Implementación de la acción de vacunar a un paciente, necesita como parámetro una vacuna que no esté vacía, en base a los valores de esta procede a eliminarla de la lista de vacunas pendientes. En la actualidad no comprueba la coherencia con el estado de vacunación del paciente y esto podría ser una opción válida a implementar en el futuro. actual versión (0.1)

trabajarVacuna

```
public void trabajarVacuna()
```

Trabaja la lista de trabajos de vacunación por completo, comprueba la posibilidad de vacunar antes de proceder

realizar_Prueba

```
public void realizar_Prueba(Pruebas pruebaApaciente)
```

Método que formaliza la acción de realizar la prueba, cambia el estado de la prueba a ejecutado, podría haber sido sobrecargado para aceptar polimorfismo con respecto al parámetro (al trabajador) pero por semántica me ha parecido que era mejor mantener los métodos separados, el de realizar la prueba en sí y el de analizarla. El método está sobreescrito en cada subclase y se accede al apropiado en tiempo de ejecución. versión (0.1)

Parameters:

pruebaApaciente - una prueba del enfermero

enfermero - el segundo parámetro se contempla en el método de la prueba. requiere de enfermero como parámetro para poder realizar la prueba.

remove_Tratamiento

```
public void remove_Tratamiento(Tratamiento x)
```

metodo sobrecargado de la clase padre persona. Como el enfermero tiene una lista de vacunas es necesaria la posibilidad de borrar los elementos de esta lista tambien y de mandar una alerta al modulo de vacunacion para buscar la ficha en este modulo y poder eliminarla acordemente. En esta version el modulo de vacunacion no esta completamente terminado. version (0.1)

Overrides:

`remove_Tratamiento` in class `Person`

Parameters:

x - tratamiento

trabajarPrueba

```
public void trabajarPrueba()
```

metodo que hace trabajar al enfermero en su lista de trabajos semanal

add_Tratamiento

```
public void add_Tratamiento(Tratamiento x)
```

Metodo sobrecargado de la clase padre Trabajador. El enfermero tiene dos colas de tratamientos, una para las vacunas y otra para el plan semanal. Este metodo deberia de lanzar una excepcion que pueda ser manejada por la funcion llamante en caso de que el enfermero rechaze la orden En la version actual esta funcion no esta implementada. Deberia de hacerse las pertinentes comprobaciones antes de hacer uso de esta funcion.

Overrides:

`add_Tratamiento` in class `Person`

Parameters:

x - tratamiento a aceptar.

planSemanal_espacioLibre

```
public boolean planSemanal_espacioLibre()
```

metodo sobreescrito de la funcion padre, aplica las restricciones pertinentes a la clase enfermero, valor almacenado en clase estatico

Specified by:

`planSemanal_espacioLibre` in class `Trabajador`

planSemanal_espacioLibre

public boolean planSemanal_espacioLibre(int x)

metodo sobreescrito, aplica una restriccion diferente a los limites de trabajo

Overrides:

planSemanal_espacioLibre in class Trabajador

Parameters:

x - de tratamientos por semana

Returns:

puede aceptar otro tratamiento.

aceptar_Tratamiento

public void aceptar_Tratamiento(Tratamiento x)

metodo sobreescrito de la clase padre, hace comprobaciones extra y redirige el tratamiento a la lista adecuada

Specified by:

aceptar_Tratamiento in class Trabajador

PACKAGE **CLASS** TREE INDEX HELP

ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class estatico

java.lang.Object
estatico

```
public abstract class estatico
extends Object
```

Clase que almacena funciones, metodos y parametros de utilidad general en el programa. Permite centralizar y parametrizar el programa de acorde a las exigencias que se puedan presentar, como un cambio en el tiempo entre vacunas

Version:

(0.1)

Author:

(Francisco Sanchez)

Field Summary

Fields

Modifier and Type	Field	Description
(package private) static int	ENF_LIMITE	
(package private) static int	NUSUARIOTEST	
(package private) static int	PCR	
(package private) static int	PROBABILIDADNEGATIVO	
(package private) static int	SEROLOGICO	
(package private) static int	TEC_LIMITE	
(package private) static int	TIEMPOVACUNAS	

Method Summary

All Methods

Static Methods

Concrete Methods

Modifier Method and Type	Description
static carga_bbdd_prueba(Clinica ClinicaSanPedro) BBDD	metodo de apoyo para el modo de test (o debug) Carga usuarios en una base

```
static  numeroRandom(int ventana)
int
```

```
static  pedirConfirmacion()
boolean
```

```
static  pedirNumero ()
int
```

```
static  pedirNumero(int inicio, int ultimo)
int
```

```
static  pedirTexto()
String
```

```
static  print_imp()
void
```

```
static  resultadoAnticuerpos()
int
```

```
static  resultadoTest()
boolean
```

de datos, se apoya de las funciones de generacion de numeros aleatorios de la clase estatico

generador de numeros aleatorios

Pide una confirmacion al usuario, devuelve true o false.

Funcion que pide una secuencia numerica a un usuario y la suministra al programa, esta version no tiene restriccion en cuanto rango de numeros, mientras sea un numero.

Funcion que pide una secuencia numerica a un usuario y la suministra al programa, esta version tiene restriccion en cuanto rango de numeros, primer valor representa el inicio y el segundo el final.

Pide una cadena de caracteres al usuario para suministrarla en otros puntos de programa, como en nombres o apellidos.

imprime un marco para la visualizacion posterior de datos persona

Devuelve el resultado de anticuerpos en base a 100

funcion que devuelve un resultado positivo o negativo en base a un numero random y a 3 posibilidades entre 10, representando la probabilidad que tienes de dar positivo en el test

Methods inherited from class java.lang.Object

```
clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Field Detail

TIEMPOVACUNAS

```
static final int TIEMPOVACUNAS
```

See Also:

[Constant Field Values](#)

PCR

```
static final int PCR
```

See Also:

[Constant Field Values](#)

SEROLOGICO

```
static final int SEROLOGICO
```

See Also:

[Constant Field Values](#)

NUSUARIOTEST

```
static final int NUSUARIOTEST
```

See Also:

[Constant Field Values](#)

TEC_LIMITE

```
static final int TEC_LIMITE
```

See Also:

[Constant Field Values](#)

ENF_LIMITE

```
static final int ENF_LIMITE
```

See Also:

[Constant Field Values](#)

PROBABILIDADNEGATIVO

```
static final int PROBABILIDADNEGATIVO
```

See Also:

[Constant Field Values](#)

Method Detail

resultadoTest

```
public static boolean resultadoTest()
```

funcion que devuelve un resultado positivo o negativo en base a un numero random y a 3 posibilidades entre 10, representando la probabilidad que tienes de dar positivo en el test

resultadoAnticuerpos

```
public static int resultadoAnticuerpos()
```

Devuelve el resultado de anticuerpos en base a 100

Returns:

int anticuerpos

numeroRandom

```
public static int numeroRandom(int ventana)
```

generador de numeros aleatorios

Parameters:

ventana - indica la ultima cifra generable; o es posibilidad return numeroRandom

carga_bbdd_prueba

```
public static BBDDcarga_bbdd_prueba(Clinica ClinicaSanPedro)
```

metodo de apoyo para el modo de test (o debug) Carga usuarios en una base de datos, se apoya de las funciones de generacion de numeros aleatorios de la clase estatico

Returns:

baseDatos

pedirNumero

```
public static int pedirNumero()
```

Funcion que pide una secuencia numerica a un usuario y la suministra al programa, esta version no tiene restriccion en cuanto rango de numeros, mientras sea un numero.

Returns:

entradaInt el numero introducido por el usuario

pedirNumero

```
public static int pedirNumero(int inicio, int ultimo)
```

Funcion que pide una secuencia numerica a un usuario y la suministra al programa, esta version tiene restriccion en cuanto rango de numeros, primer valor representa el inicio y el segundo el final.

Returns:

entradaInt el numero introducido por el usuario

print_imp

```
public static void print_imp()
```

imprime un marco para la visualizacion posterior de datos persona

pedirTexto

```
public static String pedirTexto()
```

Pide una cadena de caracteres al usuario para suministrarlala en otros puntos de programa, como en nombres o apellidos. Esta version no tiene restricciones en la actual version (0.1)

pedirConfirmacion

```
public static boolean pedirConfirmacion()
```

Pide una confirmacion al usuario, devuelve true o false.

Returns:

boolean confirmacion

PACKAGE CLASS TREE INDEX HELP

ALL CLASSES

SUMMARY: NESTED | [FIELD](#) | CONSTR | [METHOD](#) DETAIL: [FIELD](#) | CONSTR | [METHOD](#)

Class JJ

```
java.lang.Object
    Tratamiento
        Vacunas
            JJ
```

All Implemented Interfaces:

Serializable

```
public class JJ
extends Vacunas
```

Write a description of class JJ here.

Version:

(0.1)

Author:

(Francisco Sanchez)

See Also:

Serialized Form

Field Summary

Fields

Modifier and Type	Field	Description
(package private) boolean	lleno	

Constructor Summary

Constructors

Constructor	Description
JJ()	Constructor for objects of class JJ

Method Summary

All Methods	Instance Methods	Concrete Methods
-------------	------------------	------------------

Modifier and Type	Method	Description
boolean	usar ()	representa la accion de aplicar una dosis de la vacuna

Methods inherited from class Vacunas

```
cuota_tiempo, get_dosisRestantes, get_lapsoDias, vaciar
```

Methods inherited from class Tratamiento

```
aceptado, cuota_tiempo, get_dni_Paciente, get_Enfermero, get_fecha, get_Paciente,
get_Tecnico, get_tipoTratamiento, remove_Enfermero, remove_Paciente,
remove_Tecnico, remove_Tratamiento, set_enfermeroAsignado, set_paciente,
set_tecnico, timeStamp
```

Methods inherited from class java.lang.Object

```
clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Field Detail

lleno

```
boolean lleno
```

Constructor Detail

JJ

```
public JJ()
```

Constructor for objects of class JJ

Method Detail

usar

```
public boolean usar()
```

Description copied from class: Vacunas

representa la accion de aplicar una dosis de la vacuna

Overrides:

[usar](#) in class [Vacunas](#)

Returns:

boolean si la vacuna tenia dosis suficientes devuelve true

PACKAGE **CLASS** TREE INDEX HELP

ALL CLASSES

SUMMARY: NESTED | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Class Moderna

```
java.lang.Object
    Tratamiento
        Vacunas
            Moderna
```

All Implemented Interfaces:

Serializable

```
public class Moderna
extends Vacunas
```

Write a description of class Moderna here.

Version:

(0.1)

Author:

(Francisco Sanchez)

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor	Description
Moderna ()	Constructor for objects of class Moderna

Method Summary

Methods inherited from class Vacunas

cuota_tiempo, get_dosisRestantes, get_lapsoDias, usar, vaciar

Methods inherited from class Tratamiento

aceptado, cuota_tiempo, get_dni_Paciente, get_Enfermero, get_fecha, get_Paciente, get_Tecnico, get_tipoTratamiento, remove_Enfermero, remove_Paciente,

```
remove_Tecnico, remove_Tratamiento, set_enfermeroAsignado, set_paciente,  
set_tecnico, timeStamp
```

Methods inherited from class java.lang.Object

```
clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Constructor Detail

Moderna

```
public Moderna()
```

Constructor for objects of class Moderna

PACKAGE [CLASS](#) TREE INDEX HELP

ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class ModuloConfinamiento

java.lang.Object
ModuloConfinamiento

```
public class ModuloConfinamiento
extends Object
```

Esta clase se encarga de los confinamientos, utiliza una clase privada NodoConfinamiento, compuesta de 3 listas las cuales van pasandose las pruebas segun se confirme que el paciente esta aislandose. Puede extenderse con una funcion que alerte a la policia despues de tres intentos fallidos de comunicacion o con la inclusion de comentarios en las pruebas/casos con lo que se podria utilizar un patron decorador antes de incluir la prueba en el nodo, sustituyendo estas listas de pruebas por esta clase decoradora. En esta version no estan implementadas estas opciones extra. Este modulo tiene una lista que va recibiendo alertas y las almacena en una lista para ser procesadas, la funcion que las procesa comprueba que el usuario de la prueba alerta no este ya en el sistema ya que los antigenos se pueden repetir sin restriccion de tiempo. Si la alerta la a disparado un test serologico (error) lo descarta. Al procesar satisfactoriamente una alerta la coloca en un nodo "hoy" listo para ser procesado al final del dia. Al finalizar el dia se recorre la lista de nodos, se reinicia la logica de las listas internas (procesar, no contestado y confirmado y se eliminan los nodos con una antiguedad mayor de 10 dias con respecto a la fecha actual. Estos nodos se almacenan en una lista de nodos para pedir analisis serologicos post confinamiento. A continuacion se almacena el nodo "hoy" al principio de la lista.

Version:

(0.1)

Author:

(Francisco Sanchez)

Field Summary

Fields

Modifier and Type	Field	Description
(package private) <code>LinkedList<Pruebas></code>	<code>alertas</code>	
(package private) <code>Clinica</code>	<code>clinica</code>	
(package private) <code>LinkedList<ModuloConfinamiento.NodoConfinamiento></code>	<code>confinamiento</code>	
(package private) <code>ModuloConfinamiento.NodoConfinamiento</code>	<code>hoy</code>	
(package private) <code>LinkedList<ModuloConfinamiento.NodoConfinamiento></code>	<code>pendienteSerologico</code>	

Constructor Summary

Constructors

Constructor	Description
<code>ModuloConfinamiento(Clinica clinica)</code>	Constructor de la clase clinica

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type	Method	Description
void	<code>alerta(Pruebas prueba)</code>	agrega una alerta a la lista de alertas
boolean	<code>contiene_Usuario(Paciente y)</code>	recorre las listas buscando al usuario pasado por parametro
void	<code>finalizar_dia()</code>	metodo para finalizar el dia, agrega el nodo hoy a la lista y elimina los nodos con una antiguedad superior a diez dias.
void	<code>finalizar_dia(int diaPrueba)</code>	metodo debug, acorta el confinamiento por un valor de dias igual al valor pasado por parametro.
void	<code>imprimir_numeroAlertas()</code>	imprime el numero de alertas almacenadas en la lista a la espera de ser procesadas
void	<code>imprimir_resumen()</code>	imprime un resumen de los dias de confinamiento
void	<code>imprimir_serologico()</code>	imprime un resumen de los nodos en la cola de finalizados de confinamiento; los que esperan por un analisis serologico.
void	<code>ordenaListaNodoConfinamiento()</code>	metodo debug, no tiene sentido en la implementacion final del programa pues los nodos siempre se deberian de incluir en orden cronologico.
void	<code>pedir_Serologico(Administrador administrator)</code>	metodo para pedir analisis serologicos desde el modulo de confinamiento.

```
void    procesar_alertas()
```

procesa la lista de alertas

```
void    set_fecha_nodo_hoy(int dias)
```

metodo debug, cambia la fecha del modulo hoy por el valor indicado por parametro un valor negativo cambia la fecha del nodo en el pasado, un valor positivo lo parametriza en el futuro.

Methods inherited from class java.lang.Object

```
clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Field Detail

alertas

LinkedList<Pruebas> alertas

confinamiento

LinkedList<ModuloConfinamiento.NodoConfinamiento> confinamiento

hoy

ModuloConfinamiento.NodoConfinamiento hoy

pendienteSerologico

LinkedList<ModuloConfinamiento.NodoConfinamiento> pendienteSerologico

clinica

Clinica clinica

Constructor Detail

ModuloConfinamiento

ModuloConfinamiento(Clinica clinica)

Constructor de la clase clinica

Method Detail

alerta

public void alerta(Pruebas prueba)

agrega una alerta a la lista de alertas

Parameters:

prueba -

procesar_alertas

public void procesar_alertas()

procesa la lista de alertas

contiene_Usuario

public boolean contiene_Usuario(Paciente y)

recorre las listas buscando al usuario pasado por parametro

Returns:

encontrado si ha encontrado al usuario en las listas.

pedir_Serologico

public void pedir_Serologico(Administrador administrator)

metodo para pedir analisis serologicos desde el modulo de confinamiento. procesa la lista al completo

Parameters:

administrador - de la clinica

finalizar_dia

```
public void finalizar_dia()
```

metodo para finalizar el dia, agrega el nodo hoy a la lista y elimina los nodos con una antiguedad superior a diez dias.

finalizar_dia

```
public void finalizar_dia(int diaPrueba)
```

metodo debug, acorta el confinamiento por un valor de dias igual al valor pasado por parametro. Util para pruebas.

imprimir_resumen

```
public void imprimir_resumen()
```

imprime un resumen de los dias de confinamiento

imprimir_serologico

```
public void imprimir_serologico()
```

imprime un resumen de los nodos en la cola de finalizados de confinamiento; los que esperan por un analisis serologico.

imprimir_numeroAlertas

```
public void imprimir_numeroAlertas()
```

imprime el numero de alertas almacenadas en la lista a la espera de ser procesadas

set_fecha_nodo_hoy

```
public void set_fecha_nodo_hoy(int dias)
```

metodo debug, cambia la fecha del modulo hoy por el valor indicado por parametro un valor negativo cambia la fecha del nodo en el pasado, un valor positivo lo parametriza en el futuro.

Parameters:

dias -

ordenaListaNodoConfinamiento

```
public void ordenaListaNodoConfinamiento()
```

metodo debug, no tiene sentido en la implementacion final del programa pues los nodos siempre se deberian de incluir en orden cronologico.

[PACKAGE](#) [CLASS](#) [TREE](#) [INDEX](#) [HELP](#)[ALL CLASSES](#)[SUMMARY: NESTED | FIELD | CONSTR | METHOD](#) [DETAIL: FIELD | CONSTR | METHOD](#)

Class ModuloVacunacion

java.lang.Object
ModuloVacunacion

```
public class ModuloVacunacion
extends Object
```

Write a description of class ModuloVacunacion here.

Version:

(a version number or a date)

Author:

(your name)

Constructor Summary

Constructors

Constructor	Description
<code>ModuloVacunacion()</code>	Constructor for objects of class ModuloVacunacion

Method Summary

All Methods **Instance Methods** **Concrete Methods**

Modifier and Type	Method	Description
int	<code>sampleMethod(int y)</code>	An example of a method - replace this comment with your own

Methods inherited from class java.lang.Object

`clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

ModuloVacunacion

```
public ModuloVacunacion()
```

Constructor for objects of class ModuloVacunacion

Method Detail**sampleMethod**

```
public int sampleMethod(int y)
```

An example of a method - replace this comment with your own

Parameters:

y - a sample parameter for a method

Returns:

the sum of x and y

PACKAGE **CLASS** TREE INDEX HELP

ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class Nevera

java.lang.Object
Nevera

```
public class Nevera
extends Object
```

Clase que representa una nevera donde se almacenan las vacunas

Version:

(0.1)

Author:

(Francisco Sanchez)

Constructor Summary

Constructors

Constructor	Description
<code>Nevera ()</code>	Constructor de la clase nevera, inicializa los valores a 0

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type	Method	Description
int	<code>get_numeroJJ ()</code>	metodo que devuelve el numero de vacunas en total de un tipo dado
int	<code>get_numeroModerna ()</code>	metodo que devuelve el numero de vacunas en total de un tipo dado
int	<code>get_numeroPfizer ()</code>	metodo que devuelve el numero de vacunas en total de un tipo dado
int	<code>get_numeroVacunas ()</code>	metodo que devuelve el numero de vacunas en total almacenadas en la nevera
Vacunas	<code>get_vacuna_Random ()</code>	funcion que devuelve una vacuna de forma aleatoria de la nevera, si la que se ha elegido no tiene existencias se vuelve a elejir aleatoriamente, hasta que al final devuelve una.
JJ	<code>get_vacunaJJ ()</code>	metodo que devuelve una vacuna especifica, de un tipo dado

<code>Moderna get_vacunaModerna()</code>	metodo que devuelve una vacuna especifica, de un tipo dado
<code>Pfizer get_vacunaPfizer()</code>	metodo que devuelve una vacuna especifica, de un tipo dado
<code>void recibe_JJ(int x)</code>	metodos que devuelven una vacuna de forma no random.
<code>void recibe_Moderna(int x)</code>	
<code>void recibe_Pfizer(int x)</code>	

Methods inherited from class java.lang.Object

`clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

Nevera

`public Nevera()`

Constructor de la clase nevera, inicializa los valores a 0

Method Detail

get_vacuna_Random

`public Vacunas get_vacuna_Random()`

funcion que devuelve una vacuna de forma aleatoria de la nevera, si la que se ha elegido no tiene existencias se vuelve a elejir aleatoriamente, hasta que al final devuelve una.

Returns:

vacunita vacuna de forma aleatoria

recibe_JJ

`public void recibir_JJ(int x)`

metodos que devuelven una vacuna de forma no random. No util para el programa principal, pero si para las pruebas.

recibe_Pfizer

```
public void recibe_Pfizer(int x)
```

recibe_Moderna

```
public void recibe_Moderna(int x)
```

get_numeroVacunas

```
public int get_numeroVacunas()
```

metodo que devuelve el numero de vacunas en total almacenadas en la nevera

Returns:

vacunasTotal

get_numeroPfizer

```
public int get_numeroPfizer()
```

metodo que devuelve el numero de vacunas en total de un tipo dado

Returns:

vacunasPfizer

get_numeroModerna

```
public int get_numeroModerna()
```

metodo que devuelve el numero de vacunas en total de un tipo dado

Returns:

vacunasModerna

get_numeroJJ

```
public int get_numeroJJ()
```

metodo que devuelve el numero de vacunas en total de un tipo dado

Returns:

vacunasJJ

get_vacunaJJ

public JJ get_vacunaJJ()

metodo que devuelve una vacuna especifica, de un tipo dado

Returns:

JJ

get_vacunaPfizer

public Pfizer get_vacunaPfizer()

metodo que devuelve una vacuna especifica, de un tipo dado

Returns:

Pfizer

get_vacunaModerna

public Moderna get_vacunaModerna()

metodo que devuelve una vacuna especifica, de un tipo dado

Returns:

Moderna

[PACKAGE](#) [CLASS](#) [TREE](#) [INDEX](#) [HELP](#)[ALL CLASSES](#)[SUMMARY: NESTED | FIELD | CONSTR | METHOD](#) [DETAIL: FIELD | CONSTR | METHOD](#)

Class Paciente

```
java.lang.Object
  Person
    Paciente
```

All Implemented Interfaces:

[Serializable](#)

```
public class Paciente
extends Person
```

Paciente es una subclase de Person, se construye con un digito dni que representa un valor unico, consta de dos atributos extras con respecto a la clase persona, primeraDosis que representa la situacion de que el paciente haya recibido una dosis de dos y vacunado, que representa la totalidad de la vacuacion. Si se administra una vacuna de 1 sola dosis se validan los dos. No es necesaria la diferenciacion

Version:

(0.1)

Author:

(Francisco Sanchez)

See Also:

[Serialized Form](#)

Constructor Summary

Constructors

Constructor	Description
Paciente(int DNI)	Constructor de la clase paciente
Paciente(int DNI, String nombre, String apellido1, String apellido2, int edad)	Constructor de la clase paciente con atributos directos, no necesario en el programa final pero util para hacer pruebas.

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier	Method and Type	Description
----------	-----------------	-------------

<code>boolean get_estadoPrimeraDosis()</code>	confirma que haya recibido su primera dosis de vacuna
<code>boolean get_estadoVacunacion()</code>	confirma que haya recibido la totalidad de la vacunacion
<code>String nomYapell()</code>	Devuelve una cadena de caracteres con el nombre y apellido de la persona representada en el objeto
<code>boolean puede_AnalisisSerologico()</code>	Comprueba que el paciente pueda someterse a un analisis serologico
<code>boolean puede_entrePruebas(int seleccion)</code>	El metodo ejecutor de las comprobaciones para las pruebas, hace uso del polimorfismo para evitar errores en tiempo de ejecucion.
<code>boolean puede_TestPcr()</code>	Comprueba que el paciente pueda someterse a un test PCR
<code>boolean puede_Vacuna()</code>	Comprueba que el paciente pueda someterse a una vacunacion; primera o segunda dosis.
<code>void vacunar(Vacunas x)</code>	Administra una dosis de vacuna al objeto paciente, altera el estado del objeto vacuna y lo anhade a la cola de tratamientos si es la primera vez que se usa en el paciente.

Methods inherited from class Person

```
add_Tratamiento, contains_Tratamiento, formularioNuevoRegistro, get_apellido1,
get_apellido2, get_edad, get_identificadorDNI, get_nombre, get_tratamientos,
imprimir_Tratamientos, nueva_listaTratamientos, numero_tratamientos,
poll_Tratamiento, remove_Tratamiento, set_apellido1, set_apellido2, set_edad,
set_nombre, toString
```

Methods inherited from class java.lang.Object

```
clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait
```

Constructor Detail

Paciente

```
public Paciente(int DNI)
```

Constructor de la clase paciente

Paciente

```
public Paciente(int DNI,  
               String nombre,  
               String apellido1,  
               String apellido2,  
               int edad)
```

Constructor de la clase paciente con atributos directos, no necesario en el programa final pero util para hacer pruebas.

Method Detail

nomYapell

```
public String nomYapell()
```

Devuelve una cadena de caracteres con el nombre y apellido de la persona representada en el objeto

Returns:

String nombre y apellido del paciente

vacunar

```
public void vacunar(Vacunas x)
```

Administra una dosis de vacuna al objeto paciente, altera el estado del objeto vacuna y lo anhade a la cola de tratamientos si es la primera vez que se usa en el paciente.

get_estadoPrimeraDosis

```
public boolean get_estadoPrimeraDosis()
```

confirma que haya recibido su primera dosis de vacuna

Returns:

primeraDosis ha recibido primera dosis

get_estadoVacunacion

```
public boolean get_estadoVacunacion()
```

confirma que haya recibido la totalidad de la vacunacion

Returns:

vacunado el estado de vacunacion del usuario

puede_AnalisisSerologico

```
public boolean puede_AnalisisSerologico()
```

Comprueba que el paciente pueda someterse a un analisis serologico

Returns:

boolean

puede_TestPcr

```
public boolean puede_TestPcr()
```

Comprueba que el paciente pueda someterse a un test PCR

Returns:

boolean

puede_Vacuna

```
public boolean puede_Vacuna()
```

Comprueba que el paciente pueda someterse a una vacunacion; primera o segunda dosis.

Returns:

boolean

puede_entrePruebas

```
public boolean puede_entrePruebas(int seleccion)
```

El metodo ejecutor de las comprobaciones para las pruebas, hace uso del polimorfismo para evitar errores en tiempo de ejecucion. Comprueba el tiempo habilitante entre pruebas de una variable estatica en una clase concreta, definidora de parametros. imprime la informacion relativa a los dias restantes para la prueba en caso de que no se pueda repetir en el mismo dia

Class Person

`java.lang.Object`
Person

All Implemented Interfaces:

`Serializable`

Direct Known Subclasses:

`Paciente, Trabajador`

```
public abstract class Person
extends Object
implements Serializable
```

Abstract class Person - Superclase, define los atributos de un objeto abstracto persona Atributos principales designados en la clase: Identificador Nombre Apellido Apellido Edad lista de tratamientos (el significado de esta lista varia en función de la subclase)

Version:

(0.1)

Author:

(Francisco Sanchez)

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor	Description
<code>Person(int DNI)</code>	clase constructora, define solo el dni, que pasa a ser un atributo inmutable

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type	Method	Description
<code>void</code>	<code>add_Tratamiento(Tratamiento x)</code>	mutador, agrega un tratamiento a la lista de

		tratamientos
boolean	<code>contains_Tratamiento(Tratamiento x)</code>	accesor, comprueba si el tratamiento dado como parametro esta contenido en la lista de tratamientos de la persona
void	<code>formularioNuevoRegistro()</code>	funcion mutadora, pide al usuario que introduzca los datos de la nueva persona creada
<code>String</code>	<code>get_apellido1()</code>	accesor
<code>String</code>	<code>get_apellido2()</code>	accesor
int	<code>get_edad()</code>	accesor
int	<code>get_identificadorDNI()</code>	accesor
<code>String</code>	<code>get_nombre()</code>	accesor
<code>LinkedList<Tratamiento></code>	<code>get_tratamientos()</code>	devuelve la lista de tratamientos
void	<code>imprimir_Tratamientos()</code>	funcion que imprime los datos almacenados en la lista de tratamientos
void	<code>nueva_listaTratamientos()</code>	mutador, asigna una nueva lista de tratamientos
int	<code>numero_tratamientos()</code>	accesor, devuelve el valor de tamaño de la lista tratamientos
<code>Tratamiento</code>	<code>poll_Tratamiento()</code>	elimina y devuelve un elemento del principio de la lista de tratamientos
void	<code>remove_Tratamiento(Tratamiento x)</code>	mutador, elimina un tratamiento de la lista de tratamientos
void	<code>set_apellido1(String apellido)</code>	mutador, asigna apellido
void	<code>set_apellido2(String apellido)</code>	mutador, asigna segundo apellido
void	<code>set_edad(int edad)</code>	mutador, asigna edad
void	<code>set_nombre(String nombre)</code>	mutador, asigna nombre
<code>String</code>	<code>toString()</code>	accesor, sobreescribe la funcion <code>toString</code> devuelve

una cadena de los atributos de la persona

Methods inherited from class java.lang.Object

clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Person

Person(int DNI)

clase constructora, define solo el dni, que pasa a ser un atributo inmutable

Method Detail

get_identificadorDNI

public int get_identificadorDNI()

accesor

Returns:

identificadorDNI

get_nombre

public String get_nombre()

accesor

Returns:

nombre

get_apellido1

public String get_apellido1()

accesor

Returns:

apellido1

get_apellido2

public String get_apellido2()

accesor

Returns:

apellido2

get_edad

public int get_edad()

accesor

Returns:

edad

nueva_listaTratamientos

public void nueva_listaTratamientos()

mutador, asigna una nueva lista de tratamientos

set_nombre

public void set_nombre(String nombre)

mutador, asigna nombre

Parameters:

nombre -

set_apellido1

public void set_apellido1(String apellido)

mutador, asigna apellido

Parameters:

apellido -

set_apellido2

public void set_apellido2(String apellido)

mutador, asigna segundo apellido

Parameters:

apellido -

set_edad

public void set_edad(int edad)

mutador, asigna edad

Parameters:

edad -

toString

public String toString()

accesor, sobreescribe la funcion toString devuelve una cadena de los atributos de la persona

Overrides:

[toString](#) in class [Object](#)

formularioNuevoRegistro

public void formularioNuevoRegistro()

funcion mutadora, pide al usuario que introduzca los datos de la nueva persona creada

get_tratamientos

public LinkedList <Tratamiento> get_tratamientos()

devuelve la lista de tratamientos

Returns:

tratamientos

poll_Tratamiento

public Tratamiento poll_Tratamiento()

elimina y devuelve un elemento del principio de la lista de tratamientos

Returns:

Tratamiento

add_Tratamiento

public void add_Tratamiento(Tratamiento x)

mutador, agrega un tratamiento a la lista de tratamientos

Parameters:

x - tratamiento

numero_tratamientos

public int numero_tratamientos()

accesor, devuelve el valor de tamano de la lista tratamientos

Returns:

size

remove_Tratamiento

public void remove_Tratamiento(Tratamiento x)

mutador, elimina un tratamiento de la lista de tratamientos

Parameters:

x - tratamiento

contains_Tratamiento

public boolean contains_Tratamiento(Tratamiento x)

accesor, comprueba si el tratamiento dado como parametro esta contenido en la lista de tratamientos de la persona

Parameters:

x - tratamiento

Returns:

boolean, contiene

imprimir_Tratamientos

```
public void imprimir_Tratamientos()
```

funcion que imprime los datos almacenados en la lista de tratamientos

PACKAGE **CLASS** TREE INDEX HELP

ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Hierarchy For Package <Unnamed>

Class Hierarchy

- [java.lang.Object](#)
 - [BBDD](#) (implements `java.io.Serializable`)
 - [Clinica](#)
 - [estatico](#)
 - [ModuloConfinamiento](#)
 - [ModuloVacunacion](#)
 - [Nevera](#)
 - [Person](#) (implements `java.io.Serializable`)
 - [Paciente](#)
 - [Trabajador](#)
 - [Administrador](#)
 - [Enfermero](#)
 - [Tecnico](#)
- [Tratamiento](#) (implements `java.io.Serializable`)
 - [Pruebas](#)
 - [AnalisisSerologico](#)
 - [Antigenos](#)
 - [PruebaClasica](#)
 - [PruebaRapida](#)
 - [TestPcr](#)
 - [Vacunas](#)
 - [JJ](#)
 - [Moderna](#)
 - [Pfizer](#)

A B C E F G H I J L M N O P R S T U V

All Classes All Packages

A**aceptado()** - Method in class **Tratamiento**

metodo que comprueba si este tratamiento ha sido correctamente asignado.

aceptar_Tratamiento(Tratamiento) - Method in class **Administrador****aceptar_Tratamiento(Tratamiento)** - Method in class **Enfermero**

metodo sobreescrito de la clase padre, hace comprobaciones extra y redirige el tratamiento a la lista adecuada

aceptar_Tratamiento(Tratamiento) - Method in class **Tecnico**

metodo sobreescrito del metodo abstracto de la clase trabajador.

aceptar_Tratamiento(Tratamiento) - Method in class **Trabajador**

metodos interfaz para poder usar polimorfismo, sobreescritos en clases hijas

actualizar_stockVacunas(Administrador) - Method in class **Clinica**

Metodo que actualiza el stock de vacunas, necesario que el usuario de la clinica en el momento sea un administrador (medida de seguridad y bloqueo)

add_registro(int, Person) - Method in class **BBDD****add_Tratamiento(Tratamiento)** - Method in class **Administrador****add_Tratamiento(Tratamiento)** - Method in class **BBDD****add_Tratamiento(Tratamiento)** - Method in class **Enfermero**

Metodo sobrecargado de la clase padre Trabajador.

add_Tratamiento(Tratamiento) - Method in class **Person**

mutador, agrega un tratamiento a la lista de tratamientos

add_Tratamiento(Tratamiento) - Method in class **Tecnico**

metodo sobreescrito para agregar tratamientos a la lista con filtro.

Administrador - Class in <Unnamed>

Esta clase representa al administrador de la clinica, las funciones del administrador estan declaradas en la clinica y esta comprueba que el usuario es un administrador para permitir la manipulacion de esta.

Administrador(int, Clinica) - Constructor for class **Administrador****Administrador(Clinica, int, String, String, String, int)** - Constructor for class **Administrador****agrega_JJ(int)** - Method in class **BBDD****agrega_Moderna(int)** - Method in class **BBDD****agrega_Pfizer(int)** - Method in class **BBDD****agregar_planSemanal(Pruebas)** - Method in class **Trabajador**

agrega una prueba al plan semanal

alerta(Pruebas) - Method in class **ModuloConfinamiento**

agrega una alerta a la lista de alertas

alerta(Tecnico, Pruebas) - Method in class **Clinica**

Manda la prueba con resultado positivo al modulo de confinamiento.

alerta_Clinica(Trabajador, Tratamiento) - Method in class **Trabajador**

envia una senhal de alerta a la clinica, comprueba por tiempo de ejecucion que tipo de trabajador esta dando

la alerta y elije la senhal de acuerdo a sus limitaciones.

alertas - Variable in class `ModuloConfinamiento`

alertaVacuna(Trabajador, Tratamiento, boolean) - Method in class `Clinica`

Manda una senhal al modulo de vacunacion, si el parametro booleano es igual a true la senhal es de eliminacion de informacion, si la senhal es false manda una senhal al modulo para indicarle que el usuario ha sido ya vacunado y sacarlo de las colas de prioridad correspondientes.

altaUsuario(Administrador, int) - Method in class `Clinica`

Metodo que crea usuarios por eleccion

AnalisisSerologico - Class in <Unnamed>

Analisis serologico hereda de prueba aunque es un poco mas especial que el resto, esta prueba no determina si se tiene el virus, si no si se hayan presentes en el organismo los anticuerpos para este.

AnalisisSerologico() - Constructor for class `AnalisisSerologico`

constructor del analisis serologico. el constructor llama a super() automaticamente.

analizar_prueba(Tecnico) - Method in class `AnalisisSerologico`

metodo que sobreescribe al metodo de la clase padre analizar_prueba.

analizar_prueba(Tecnico) - Method in class `Pruebas`

mutador de resultado, la contraparte del metodo hacer_prueba para el tecnico.

Antigenos - Class in <Unnamed>

Abstract class Antigenos - clase abstracta, representa el comun de las pruebas de antigenos, las cuales se pueden hacer diariamente.

Antigenos() - Constructor for class `Antigenos`

asignar_prueba(int, Paciente, Administrador) - Method in class `Clinica`

metodo que asigna pruebas a los diferentes diferentes usuarios,

asignarTratamiento(Administrador) - Method in class `Clinica`

Metodo decorador del metodo asignar_prueba(), hace las veces de comprobador de estado de la clinica y configurador de parametros, tiene algunos metodos bloqueados a la espera de ser implementados (version 0.1)

asignarTratamiento(Administrador, int, int) - Method in class `Clinica`

metodo que asigna un tratamiento de forma directa, sin pedir parametros al usuario.

B

BBDD - Class in <Unnamed>

La base de datos de la que se nutre la clase clinica y donde se almacena toda la informacion, implementa la interfaz Serializable para poder hacer una copia y mandarlo como stream de digitos binarios a un archivo fuera del entorno de bluej y a la inversa, para cargar un objeto base de datos de un archivo independiente y restaurar la informacion de una ejecucion anterior.

BBDD() - Constructor for class `BBDD`

Constructor for objects of class BBDD

C

carga_bbdd_prueba(Clinica) - Static method in class `estatico`

metodo de apoyo para el modo de test (o debug) Carga usuarios en una base de datos, se apoya de las funciones de generacion de numeros aleatorios de la clase estatico

clinica - Variable in class `ModuloConfinamiento`

Clinica - Class in <Unnamed>

Clase principal del programa, desde aqui se administra y se consultan los estados de este.

Clinica() - Constructor for class `Clinica`

Constructor for objects of class Clinica

coger_prueba() - Method in class Trabajador

busca en la lista el elemento que necesita ser, o bien ejecutado por el enfermero o bien procesado por el tecnico y lo devuelve.

confinamiento - Variable in class ModuloConfinamiento**contains_registro(int)** - Method in class BBDD**contains_Tratamiento(Tratamiento)** - Method in class Person

accesor, comprueba si el tratamiento dado como parametro esta contenido en la lista de tratamientos de la persona

contiene_Usuario(Paciente) - Method in class ModuloConfinamiento

recorre las listas buscando al usuario pasado por parametro

cuota_tiempo() - Method in class Vacunas

Comprueba, accediendo a el metodo de su clase padre si se podria realizar una segunda vacuna en el dia de la comprobacion.

cuota_tiempo(int) - Method in class Tratamiento

funcion que calcula cuantos dias han pasado desde una fecha hasta la fecha almacenada en el tratamiento.

E**eliminar_registro(int)** - Method in class BBDD**ENF_LIMITE** - Static variable in class estatico**Enfermero** - Class in <Unnamed>**Enfermero(int, Clinica)** - Constructor for class Enfermero

Constructor de la clase enfermero, requiere el identificador unico el la clinica como lugar de trabajo

Enfermero(Clinica, int, String, String, String, int) - Constructor for class Enfermero

Constructor sobrecargado para crear un enfermero con todos los datos suministrado por parametro, sin uso en el programa final pero util para las opciones de debug y pruebas.

estatico - Class in <Unnamed>

Clase que almacena funciones, metodos y parametros de utilidad general en el programa.

F**finalizar_dia()** - Method in class ModuloConfinamiento

metodo para finalizar el dia, agrega el nodo hoy a la lista y elimina los nodos con una antiguedad superior a diez dias.

finalizar_dia(int) - Method in class ModuloConfinamiento

metodo debug, acorta el confinamiento por un valor de dias igual al valor pasado por parametro.

formularioNuevoRegistro() - Method in class Person

funcion mutadora, pide al usuario que introduzca los datos de la nueva persona creada

G**get_apellido1()** - Method in class Person

accesor

get_apellido2() - Method in class Person

accesor

get_archivo() - Method in class BBDD**get_dni_Paciente()** - Method in class Tratamiento

metodo selector, devuelve el identificador del paciente del tratamiento

get_dosisRestantes() - Method in class Vacunas

metodo selector de dosis restantes de la vacuna

get_edad() - Method in class Person

accesor

get_ejecutado() - Method in class Pruebas

accesor ejecutado

get_Enfermero() - Method in class Tratamiento

metodo selector, devuelve enfermero del tratamiento

get_estadoPrimeraDosis() - Method in class Paciente

confirma que haya recibido su primera dosis de vacuna

get_estadoVacunacion() - Method in class Paciente

confirma que haya recibido la totalidad de la vacunacion

get_fecha() - Method in class Tratamiento

metodo selector que devuelve la fecha asignada al tratamiento

get_identificadorDNI() - Method in class Person

accesor

get_lapsoDias() - Method in class AnalisisSerologico

metodo accesror del tiempo en dias entre pruebas de tipo serologico.

get_lapsoDias() - Method in class Antigenos

implementa el metodo abstracto de la clase padre.

get_lapsoDias() - Method in class Pruebas

metodo abstracto para implementar por las clases hijas

get_lapsoDias() - Method in class TestPcr

implementa el metodo abstracto de la clase padre.

get_lapsoDias() - Method in class Tratamiento

metodo interfaz, permite la correcta utilizacion del polimorfismo

get_lapsoDias() - Method in class Vacunas

devuelve el tiempo necesario de espera entre vacunas

get_Nevera() - Method in class BBDD

get_nombre() - Method in class Person

accesor

get_numeroJJ() - Method in class BBDD

get_numeroJJ() - Method in class Nevera

metodo que devuelve el numero de vacunas en total de un tipo dado

get_numeroModerna() - Method in class BBDD

get_numeroModerna() - Method in class Nevera

metodo que devuelve el numero de vacunas en total de un tipo dado

get_numeroPfizer() - Method in class BBDD

get_numeroPfizer() - Method in class Nevera

metodo que devuelve el numero de vacunas en total de un tipo dado

get_numeroVacunas() - Method in class Nevera

metodo que devuelve el numero de vacunas en total almacenadas en la nevera

get_Paciente() - Method in class Tratamiento

metodo selector, devuelve paciente del tratamiento

get_procesado() - Method in class Pruebas

accesor procesado

get_registro(int) - Method in class BBDD

get_resultado() - Method in class Pruebas

accesor resultado

get_Tecnico() - Method in class Tratamiento

metodo selector, devuelve tecnico del tratamiento

get_tipoTratamiento() - Method in class Tratamiento

devuelve una cadena con el tipo de subclase asignable al tratamiento return tipo de clase

get_tratamientos() - Method in class Person

devuelve la lista de tratamientos

get_vacuna_Random() - Method in class BBDD

saca una vacuna aleatoria de la nevera

get_vacuna_Random() - Method in class Nevera

funcion que devuelve una vacuna de forma aleatoria de la nevera, si la que se ha elegido no tiene existencias se vuelve a elejir aleatoriamente, hasta que al final devuelve una.

get_vacunaJJ() - Method in class BBDD

get_vacunaJJ() - Method in class Nevera

metodo que devuelve una vacuna especifica, de un tipo dado

get_vacunaModerna() - Method in class BBDD

get_vacunaModerna() - Method in class Nevera

metodo que devuelve una vacuna especifica, de un tipo dado

get_vacunaPfizer() - Method in class BBDD

get_vacunaPfizer() - Method in class Nevera

metodo que devuelve una vacuna especifica, de un tipo dado

get_vacunas() - Method in class BBDD

H

hacer_prueba(Enfermero) - Method in class Pruebas

mutador de ejecutado.

hoy - Variable in class ModuloConfinamiento

I

imprimir_completados() - Method in class Trabajador

imprime los tratamientos almacenados en la lista Tratamientos declarada en la clase persona.

imprimir_datosUsuarioUnitario() - Method in class Clinica

metodo que imprime los datos de un usuario concreto, lo implementa el menu de trabajador y el de administrador, comprueba tipo de objeto en tiempo de ejecucion (tipo de persona) e imprime la informacion especializada de la clase

imprimir_numeroAlertas() - Method in class ModuloConfinamiento

imprime el numero de alertas almacenadas en la lista a la espera de ser procesadas

imprimir_pacientes_Completados() - Method in class Trabajador

imprime la informacion de los pacientes de los tratamientos completados.

imprimir_pacientes_planSemanal() - Method in class Trabajador

imprime la informacion de los pacientes de los tratamientos del plan semanal

imprimir_planSemanal() - Method in class Trabajador

imprime el plan semanal

imprimir_resumen() - Method in class ModuloConfinamiento

imprime un resumen de los dias de confinamiento

imprimir_serologico() - Method in class ModuloConfinamiento

imprime un resumen de los nodos en la cola de finalizados de confinamiento; los que esperan por un analisis serologico.

imprimir_Tratamientos() - Method in class Person

funcion que imprime los datos almacenados en la lista de tratamientos

in_planSemanal() - Method in class Trabajador

mutador constructor de atributo planSemanal, representativo de la carga de trabajo correspondiente al trabajador

iniciar() - Method in class Clinica

Metodo principal de la clase, todas las funciones se controlan desde aqui

iterador_planSemanal() - Method in class Trabajador

iterador para la lista de plan semanal, permite recorrer y visualizar la lista sin alterar su composicion u orden.

J

JJ - Class in <Unnamed>

Write a description of class JJ here.

JJ() - Constructor for class JJ

Constructor for objects of class JJ

L

lapsoDias - Variable in class Antigenos

lista_Enfermero() - Method in class BBDD

Devuelve uan lista con los enfermeros extraidos de la lista original

lista_Pacientes() - Method in class BBDD

Devuelve una lista con los pacientes extraidos de la lista original

lista_Tecnicos() - Method in class BBDD

Devuelve una lista con los tecnicos extraidos de la lista original

lleno - Variable in class JJ

M

menuTrabajador(Trabajador) - Method in class Clinica

Menu de trabajadores, necesita que el usuario de la clinica sea un trabajador, comprueba el tipo de instancia en tiempo de ejecucion y aplica los metodos y las funciones propias de la clase final.

Moderna - Class in <Unnamed>

Write a description of class Moderna here.

Moderna() - Constructor for class Moderna

Constructor for objects of class Moderna

ModuloConfinamiento - Class in <Unnamed>

Esta clase se encarga de los confinamientos, utiliza una clase privada NodoConfinamiento, compuesta de 3 listas las cuales van pasandose las pruebas segun se confirme que el paciente esta aislandose.

ModuloConfinamiento(Clinica) - Constructor for class ModuloConfinamiento

Constructor de la clase clinica

ModuloVacunacion - Class in <Unnamed>

Write a description of class ModuloVacunacion here.

ModuloVacunacion() - Constructor for class ModuloVacunacion

Constructor for objects of class ModuloVacunacion

N**Nevera** - Class in <Unnamed>

Clase que representa una nevera donde se almacenan las vacunas

Nevera() - Constructor for class Nevera

Constructor de la clase nevera, inicializa los valores a 0

nomYapell() - Method in class Paciente

Devuelve una cadena de caracteres con el nombre y apellido de la persona representada en el objeto

nueva_listaTratamientos() - Method in class Person

mutador, asigna una nueva lista de tratamientos

num_casos_PS() - Method in class Trabajador

devuelve el tamnho de la lista del plan semanal, util para compararlo con los limites impuestos en la practica

numero_tratamientos() - Method in class Person

accesor, devuelve el valor de tamano de la lista tratamientos

numeroAdministradores - Variable in class BBDD**numeroAdministradores()** - Method in class BBDD**numeroEnfermeros** - Variable in class BBDD**numeroEnfermeros()** - Method in class BBDD**numeroPacientes** - Variable in class BBDD**numeroPacientes()** - Method in class BBDD**numeroRandom(int)** - Static method in class estatico

generador de numeros aleatorios

numeroTecnicos - Variable in class BBDD**numeroTecnicos()** - Method in class BBDD**numeroVacunas()** - Method in class BBDD

metodos accesores del objeto Nevera

NUSUARIOTEST - Static variable in class estatico**O****ordenaListaNodoConfinamiento()** - Method in class ModuloConfinamiento

metodo debug, no tiene sentido en la implementacion final del programa pues los nodos siempre se deberian de incluir en orden cronologico.

P**Paciente** - Class in <Unnamed>

Paciente es una subclase de Person, se construye con un digito dni que representa un valor unico, consta de dos atributos extras con respecto a la clase persona, primeraDosis que representa la situacion de que el paciente haya recibido una dosis de dos y vacunado, que representa la totalidad de la vacuacion.

Paciente(int) - Constructor for class Paciente

Constructor de la clase paciente

Paciente(int, String, String, String, int) - Constructor for class Paciente

Constructor de la clase paciente con atributos directos, no necesario en el programa final pero util para hacer pruebas.

PCR - Static variable in class estatico

pedir_Serologico(Administrador) - Method in class ModuloConfinamiento

metodo para pedir analisis serologicos desde el modulo de confinamiento.

pedirConfirmacion() - Static method in class estatico

Pide una confirmacion al usuario, devuelve true o false.

pedirNumero() - Static method in class estatico

Funcion que pide una secuencia numerica a un usuario y la suministra al programa, esta version no tiene restriccion en cuanto rango de numeros, mientras sea un numero.

pedirNumero(int, int) - Static method in class estatico

Funcion que pide una secuencia numerica a un usuario y la suministra al programa, esta version tiene restriccion en cuanto rango de numeros, primer valor representa el inicio y el segundo el final.

pedirTexto() - Static method in class estatico

Pide una cadena de caracteres al usuario para suministrarla en otros puntos de programa, como en nombres o apellidos.

pendienteSerologico - Variable in class ModuloConfinamiento

Person - Class in <Unnamed>

Abstract class Person - Superclase, define los atributos de un objeto abstracto persona Atributos principales designados en la clase: Identificador Nombre Apellido Apellido Edad lista de tratamientos (el significado de esta lista varia en funcion de la subclase)

Person(int) - Constructor for class Person

clase constructora, define solo el dni, que pasa a ser un atributo inmutable

Pfizer - Class in <Unnamed>

Write a description of class Pfizer here.

Pfizer() - Constructor for class Pfizer

Constructor for objects of class Pfizer

planSemanal_espacioLibre() - Method in class Administrador

planSemanal_espacioLibre() - Method in class Enfermero

metodo sobreescrito de la funcion padre, aplica las restricciones pertinentes a la clase enfermero, valor almacenado en clase estatico

planSemanal_espacioLibre() - Method in class Tecnico

comprueba que el plan semanal tenga espacio para aceptar trabajos con respecto a los limites correspondientes a la subclase

planSemanal_espacioLibre() - Method in class Trabajador

planSemanal_espacioLibre(int) - Method in class Administrador

planSemanal_espacioLibre(int) - Method in class Enfermero

metodo sobreescrito, aplica una restriccion diferente a los limites de trabajo

planSemanal_espacioLibre(int) - Method in class Tecnico

comprueba que el plan semanal tenga espacio para aceptar trabajos con respecto a los limites correspondientes a un parametro dado

planSemanal_espacioLibre(int) - Method in class Trabajador

devuelve la comprobacion de si el objeto trabajador tiene espacio en su lista de trabajos semanal.

poll_Tratamiento() - Method in class Person

elimina y devuelve un elemento del principio de la lista de tratamientos

print_imp() - Static method in class estatico

imprime un marco para la visualizacion posterior de datos persona

PROBABILIDADNEGATIVO - Static variable in class estatico

procesar_alertas() - Method in class ModuloConfinamiento

procesa la lista de alertas

PruebaClasica - Class in <Unnamed>

Esta clase solo diferencia entre tipos de pruebas de antigenos y no contiene ninguna diferencia con respecto a la clase padre

PruebaClasica() - Constructor for class PruebaClasica

Constructor for objects of class PruebaClasica

PruebaRapida - Class in <Unnamed>

Esta clase solo diferencia entre tipos de pruebas de antigenos y no contiene ninguna diferencia con respecto a la clase padre

PruebaRapida() - Constructor for class PruebaRapida

Constructor for objects of class PruebaRapida

Pruebas - Class in <Unnamed>

Abstract class Pruebas - clase de la que derivan las pruebas, esta a su vez hereda de tratamientos consta de 3 atributos mas con respecto a Tratamientos; resultado, procesado y ejecutado.

Pruebas() - Constructor for class Pruebas

puede_AnalisisSerologico() - Method in class Paciente

Comprueba que el paciente pueda someterse a un analisis serologico

puede_entrePruebas(int) - Method in class Paciente

El metodo ejecutor de las comprobaciones para las pruebas, hace uso del polimorfismo para evitar errores en tiempo de ejecucion.

puede_TestPcr() - Method in class Paciente

Comprueba que el paciente pueda someterse a un test PCR

puede_Vacuna() - Method in class Paciente

Comprueba que el paciente pueda someterse a una vacunacion; primera o segunda dosis.

R

realizar_Prueba(Pruebas) - Method in class Enfermero

Metodo que formaliza la accion de realizar la prueba, cambia el estado de la prueba a ejecutado, podria haber sido sobrecargado para aceptar polimorfismo con respecto al parametro (al trabajador) pero por semantica me ha parecido que era mejor mantener los metodos separados, el de realizar la prueba en si y el de analizarla.

recibe_JJ(int) - Method in class Nevera

metodos que devuelven una vacuna de forma no random.

recibe_Moderna(int) - Method in class Nevera

recibe_Pfizer(int) - Method in class Nevera

remove_Enfermero() - Method in class Tratamiento

metodo mutador, remueve enfermero del tratamiento

remove_Paciente() - Method in class Tratamiento

metodo mutador, remueve paciente del tratamiento

remove_Tecnico() - Method in class Tratamiento

metodo mutador, remueve tecnico del tratamiento

remove_Tratamiento() - Method in class Tratamiento

metodo que se asegura de que no queda rastro del tratamiento a lo largo del programa este metodo tiene que revisarse pues contiene bugs, version actual (0.1)

remove_Tratamiento(Tratamiento) - Method in class Enfermero

metodo sobrecargado de la clase padre persona.

remove_Tratamiento(Tratamiento) - Method in class Person

mutador, elimina un tratamiento de la lista de tratamientos

resultadoAnticuerpos() - Static method in class estatico

Devuelve el resultado de anticuerpos en base a 100

resultadoTest() - Static method in class estatico

funcion que devuelve un resultado positivo o negativo en base a un numero random y a 3 posibilidades entre

10, representando la probabilidad que tienes de dar positivo en el test

S

sampleMethod(int) - Method in class **ModuloVacunacion**

An example of a method - replace this comment with your own

SEROLOGICO - Static variable in class estatico

set_apellido1(String) - Method in class **Person**

mutador, asigna apellido

set_apellido2(String) - Method in class **Person**

mutador, asigna segundo apellido

set_edad(int) - Method in class **Person**

mutador, asigna edad

set_enfermeroAsignado(Enfermero) - Method in class **Tratamiento**

metodo mutador, asigna enfermero a la prueba

set_fecha_nodo_hoy(int) - Method in class **ModuloConfinamiento**

metodo debug, cambia la fecha del modulo hoy por el valor indicado por parametro un valor negativo cambia la fecha del nodo en el pasado, un valor positivo lo parametriza en el futuro.

set_nombre(String) - Method in class **Person**

mutador, asigna nombre

set_paciente(Paciente) - Method in class **Tratamiento**

metodo mutador, asigna paciente a la prueba

set_resultado(boolean) - Method in class **Pruebas**

mutador resultado y procesado.

set_tecnico(Tecnico) - Method in class **Tratamiento**

metodo mutador, asigna tecnico a la prueba

size_registro() - Method in class **BBDD**

T

TEC_LIMITE - Static variable in class estatico

Tecnico - Class in <Unnamed>

Funciones del tecnico: -Visualizacion de datos de los pacientes asignados -Registro y actualizacion de pruebas diagnosticas y vacunacion

Tecnico(int, Clinica) - Constructor for class **Tecnico**

constructor de la clase tecnico

Tecnico(Clinica, int, String, String, String, int) - Constructor for class **Tecnico**

constructor completo para las funciones de debug y pruebas.

TestPcr - Class in <Unnamed>

Esta clase representa a un test PCR, impone un limite en los dias entre pruebas pero no tiene nada mas especial con respecto a otra prueba cualquiera de analisis de presencia viral

TestPcr() - Constructor for class **TestPcr**

Constructor for objects of class TestPcr

TIEMPOVACUNAS - Static variable in class estatico

tiene_Trabajo() - Method in class **Trabajador**

comprueba si la cola de trabajo semanal esta vacia

timeStamp() - Method in class **Tratamiento**

metodo mutador que asigna la fecha actual al tratamiento

toString() - Method in class **Person**

accesor, sobreescribe la funcion `toString` devuelve una cadena de los atributos de la persona

Trabajador - Class in <Unnamed>

Abstract class Trabajador - write a description of the class here

Trabajador(int, Clinica) - Constructor for class Trabajador

trabajadoresLibres() - Method in class Clinica

metodo que comprueba que haya trabajadores con espacio en su plan semanal

trabajarPrueba() - Method in class Enfermero

metodo que hace trabajar al enfermero en su lista de trabajos semanal

TrabajarPruebas() - Method in class Tecnico

Pone al tecnico a trabajar, procesa todas las pruebas de a una.

trabajarVacuna() - Method in class Enfermero

Trabaja la lista de trabajos de vacunacion por completo, comprueba la posibilidad de vacunar antes de proceder

Tratamiento - Class in <Unnamed>

Abstract class Tratamiento - Esta clase representa de forma general cualquier procedimiento por el cual un paciente o usuario haya acudido a la clinica.

Tratamiento() - Constructor for class Tratamiento

constructor de la clase Tratamiento, esta es abstracta, aun asi dispone que cualquier tratamiento que se construya deba tener la fecha del momento "estampada"

U

umbral - Variable in class Clinica

usar() - Method in class JJ

usar() - Method in class Vacunas

representa la accion de aplicar una dosis de la vacuna

V

vaciar() - Method in class Vacunas

metodo mutador que pone a 0 la cantidad de dosis restantes en la vacuna

vacunacionesRestantes() - Method in class Enfermero

Devuelve el tamanno de la lista de vacunaciones

vacunar(Vacunas) - Method in class Enfermero

Implementacion de la accion de vacunar a un paciente, necesita como parametro una vacuna que no este vacia, en base a los valores de esta procede a eliminarla de la lista de vacunas pendientes.

vacunar(Vacunas) - Method in class Paciente

Administra una dosis de vacuna al objeto paciente, altera el estado del objeto vacuna y lo anhade a la cola de tratamientos si es la primera vez que se usa en el paciente.

Vacunas - Class in <Unnamed>

Abstract class Vacunas - Define unos patrones genericos para las 3 vacunas y los metodos aplicables como interfaz.

Vacunas() - Constructor for class Vacunas

constructor de la clase Vacunas, la clase es abstracta

A B C E F G H I J L M N O P R S T U V

All Classes All Packages

ALL CLASSES

How This API Document Is Organized

This API (Application Programming Interface) document has pages corresponding to the items in the navigation bar, described as follows.

Package

Each package has a page that contains a list of its classes and interfaces, with a summary for each. These pages may contain six categories:

- Interfaces
- Classes
- Enums
- Exceptions
- Errors
- Annotation Types

Class or Interface

Each class, interface, nested class and nested interface has its own separate page. Each of these pages has three sections consisting of a class/interface description, summary tables, and detailed member descriptions:

- Class Inheritance Diagram
- Direct Subclasses
- All Known Subinterfaces
- All Known Implementing Classes
- Class or Interface Declaration
- Class or Interface Description
- Nested Class Summary
- Field Summary
- Property Summary
- Constructor Summary
- Method Summary
- Field Detail
- Property Detail
- Constructor Detail
- Method Detail

Each summary entry contains the first sentence from the detailed description for that item. The summary entries are alphabetical, while the detailed descriptions are in the order they appear in the source code. This preserves the logical groupings established by the programmer.

Annotation Type

Each annotation type has its own separate page with the following sections:

- Annotation Type Declaration
- Annotation Type Description

- Required Element Summary
- Optional Element Summary
- Element Detail

Enum

Each enum has its own separate page with the following sections:

- Enum Declaration
- Enum Description
- Enum Constant Summary
- Enum Constant Detail

Tree (Class Hierarchy)

There is a [Class Hierarchy](#) page for all packages, plus a hierarchy for each package. Each hierarchy page contains a list of classes and a list of interfaces. Classes are organized by inheritance structure starting with `java.lang.Object`. Interfaces do not inherit from `java.lang.Object`.

- When viewing the Overview page, clicking on "Tree" displays the hierarchy for all packages.
- When viewing a particular package, class or interface page, clicking on "Tree" displays the hierarchy for only that package.

Index

The [Index](#) contains an alphabetic index of all classes, interfaces, constructors, methods, and fields, as well as lists of all packages and all classes.

All Classes

The [All Classes](#) link shows all classes and interfaces except non-static nested types.

Serialized Form

Each serializable or externalizable class has a description of its serialization fields and methods. This information is of interest to re-implementors, not to developers using the API. While there is no link in the navigation bar, you can get to this information by going to any serialized class and clicking "Serialized Form" in the "See also" section of the class description.

Constant Field Values

The [Constant Field Values](#) page lists the static final fields and their values.

Search

You can search for definitions of modules, packages, types, fields, methods and other terms defined in the API, using some or all of the name. "Camel-case" abbreviations are supported: for example, "InpStr" will find "InputStream" and "InputStreamReader".

This help file applies to API documentation generated by the standard doclet.

All Classes

Administrador
AnalisisSerologico
Antigenos
BBDD
Clinica
Enfermero
estatico
JJ
Moderna
ModuloConfinamiento
ModuloVacunacion
Nevera
Paciente
Person
Pfizer
PruebaClasica
PruebaRapida
Pruebas
Tecnico
TestPcr
Trabajador
Tratamiento
Vacunas

Class Pfizer

```
java.lang.Object
    Tratamiento
        Vacunas
            Pfizer
```

All Implemented Interfaces:

`Serializable`

```
public class Pfizer
extends Vacunas
```

Write a description of class Pfizer here.

Version:

(a version number or a date)

Author:

(your name)

See Also:

[Serialized Form](#)

Constructor Summary

Constructors

Constructor	Description
<code>Pfizer()</code>	Constructor for objects of class Pfizer

Method Summary

Methods inherited from class Vacunas

`cuota_tiempo, get_dosisRestantes, get_lapsoDias, usar, vaciar`

Methods inherited from class Tratamiento

`aceptado, cuota_tiempo, get_dni_Paciente, get_Enfermero, get_fecha, get_Paciente, get_Tecnico, get_tipoTratamiento, remove_Enfermero, remove_Paciente,`

```
remove_Tecnico, remove_Tratamiento, set_enfermeroAsignado, set_paciente,  
set_tecnico, timeStamp
```

Methods inherited from class java.lang.Object

```
clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Constructor Detail

Pfizer

```
public Pfizer()
```

Constructor for objects of class Pfizer

[PACKAGE](#) [CLASS](#) [TREE](#) [INDEX](#) [HELP](#)

[ALL CLASSES](#)

[SUMMARY: NESTED | FIELD | CONSTR | METHOD](#) [DETAIL: FIELD | CONSTR | METHOD](#)

Class PruebaClasica

```
java.lang.Object
    Tratamiento
        Pruebas
            Antigenos
                PruebaClasica
```

All Implemented Interfaces:

[Serializable](#)

```
public class PruebaClasica
extends Antigenos
```

Esta clase solo diferencia entre tipos de pruebas de antigenos y no contiene ninguna diferencia con respecto a la clase padre

Version:

(0.1)

Author:

(Francisco Sanchez)

See Also:

[Serialized Form](#)

Field Summary

Fields inherited from class [Antigenos](#)

`lapsoDias`

Constructor Summary

Constructors

Constructor	Description
-------------	-------------

<code>PruebaClasica()</code>	Constructor for objects of class PruebaClasica
------------------------------	--

Method Summary

Methods inherited from class Antigenos

```
get_lapsoDias
```

Methods inherited from class Pruebas

```
analizar_prueba, get_ejecutado, get_procesado, get_resultado, hacer_prueba,  
set_resultado
```

Methods inherited from class Tratamiento

```
aceptado, cuota_tiempo, get_dni_Paciente, get_Enfermero, get_fecha, get_Paciente,  
get_Tecnico, get_tipoTratamiento, remove_Enfermero, remove_Paciente,  
remove_Tecnico, remove_Tratamiento, set_enfermeroAsignado, set_paciente,  
set_tecnico, timeStamp
```

Methods inherited from class java.lang.Object

```
clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Constructor Detail

PruebaClasica

```
public PruebaClasica()
```

Constructor for objects of class PruebaClasica

PACKAGE **CLASS** TREE INDEX HELP

ALL CLASSES

SUMMARY: NESTED | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Class PruebaRapida

```
java.lang.Object
    Tratamiento
        Pruebas
            Antigenos
                PruebaRapida
```

All Implemented Interfaces:

[Serializable](#)

```
public class PruebaRapida
extends Antigenos
```

Esta clase solo diferencia entre tipos de pruebas de antigenos y no contiene ninguna diferencia con respecto a la clase padre

Version:

(0.1)

Author:

(Francisco Sanchez)

See Also:

[Serialized Form](#)

Field Summary

Fields inherited from class [Antigenos](#)

`lapsoDias`

Constructor Summary

Constructors

Constructor	Description
-------------	-------------

PruebaRapida()	Constructor for objects of class PruebaRapida
--------------------------------	---

Method Summary

Methods inherited from class Antigenos

```
get_lapsoDias
```

Methods inherited from class Pruebas

```
analizar_prueba, get_ejecutado, get_procesado, get_resultado, hacer_prueba,  
set_resultado
```

Methods inherited from class Tratamiento

```
aceptado, cuota_tiempo, get_dni_Paciente, get_Enfermero, get_fecha, get_Paciente,  
get_Tecnico, get_tipoTratamiento, remove_Enfermero, remove_Paciente,  
remove_Tecnico, remove_Tratamiento, set_enfermeroAsignado, set_paciente,  
set_tecnico, timeStamp
```

Methods inherited from class java.lang.Object

```
clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Constructor Detail

PruebaRapida

```
public PruebaRapida()
```

Constructor for objects of class PruebaRapida

PACKAGE **CLASS** TREE INDEX HELP

ALL CLASSES

SUMMARY: NESTED | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Class Pruebas

`java.lang.Object`

 Tratamiento

 Pruebas

All Implemented Interfaces:

`Serializable`

Direct Known Subclasses:

`AnalisisSerologico, Antigenos, TestPcr`

```
public abstract class Pruebas
extends Tratamiento
```

Abstract class Pruebas - clase de la que derivan las pruebas, esta a su vez hereda de tratamientos consta de 3 atributos mas con respecto a Tratamientos; resultado, procesado y ejecutado. Respectivamente muestran si ha dado un resultado positivo o negativo, si ha sido procesado por el tecnico y ejecutado por el enfermero, por lo tanto el resultado seria el ultimo atributo en asignarse y solo tendria valor si los otros dos son true.

Version:

(0.1)

Author:

(Francisco Sanchez)

See Also:

`Serialized Form`

Constructor Summary

Constructors

Constructor	Description
<code>Pruebas ()</code>	

Method Summary

All Methods

Instance Methods

Abstract Methods

Concrete Methods

Modifier and Type	Method	Description
void	<code>analizar_prueba(Tecnico tecnico)</code>	mutador de resultado, la contraparte del

metodo hacer_prueba para el tecnico.

boolean	get_ejecutado()	accesor ejecutado
abstract int	get_lapsoDias()	metodo abstracto para implementar por las clases hijas
boolean	get_procesado()	accesor procesado
boolean	get_resultado()	accesor resultado
void	hacer_prueba(Enfermero enfermero)	mutador de ejecutado.
void	set_resultado(boolean resultado)	mutador resultado y procesado.

Methods inherited from class Tratamiento

aceptado, cuota_tiempo, get_dni_Paciente, get_Enfermero, get_fecha, get_Paciente, get_Tecnico, get_tipoTratamiento, remove_Enfermero, remove_Paciente, remove_Tecnico, remove_Tratamiento, set_enfermeroAsignado, set_paciente, set_tecnico, timeStamp

Methods inherited from class java.lang.Object

clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Pruebas

```
public Pruebas()
```

Method Detail

get_lapsoDias

```
public abstract int get_lapsoDias()
```

metodo abstracto para implementar por las clases hijas

Specified by:

`get_lapsoDias` in class `Tratamiento`

Returns:

numero de dias entre pruebas

get_resultado

public boolean get_resultado()

accesor resultado

Parameters:

resultado -

set_resultado

public void set_resultado(boolean resultado)

mutador resultado y procesado.

Parameters:

resultado -

get_procesado

public boolean get_procesado()

accesor procesado

Returns:

procesado

get_ejecutado

public boolean get_ejecutado()

accesor ejecutado

Returns:

ejecutado

hacer_prueba

public void hacer_prueba(Enfermero enfermero)

mutador de ejecutado. Este es el metodo que debe usarse para cambiar el estado de una prueba, puesto que es el enfermero el que tiene que declarar que se ha ejecutado la prueba

Parameters:

enfermero - que ejecuta la prueba

analizar_prueba

```
public void analizar_prueba(Tecnico tecnico)
```

mutador de resultado, la contraparte del metodo hacer_prueba para el tecnico. el resultado deberia de aplicarse con este metodo.

Parameters:

tecnico - que realiza la prueba.

PACKAGE **CLASS** TREE INDEX HELP

ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class Tecnico

```
java.lang.Object
Person
    Trabajador
        Tecnico
```

All Implemented Interfaces:

Serializable

```
public class Tecnico
extends Trabajador
```

Funciones del tecnico: -Visualizacion de datos de los pacientes asignados -Registro y actualizacion de pruebas diagnosticas y vacunacion

Version:

(0.1)

Author:

(Francisco Sanchez)

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor	Description
<code>Tecnico(int DNI, Clinica c)</code>	constructor de la clase tecnico
<code>Tecnico(Clinica c, int DNI, String nombre, String apellido1, String apellido2, int edad)</code>	constructor completo para las funciones de debug y pruebas.

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type	Method	Description
void	<code>aceptar_Tratamiento(Tratamiento x)</code>	metodo sobreescrito del metodo abstracto de la clase trabajador.

void add_Tratamiento (Tratamiento x)	metodo sobreescrito para agregar tratamientos a la lista con filtro.
boolean planSemanal_espacioLibre ()	comprueba que el plan semanal tenga espacio para aceptar trabajos con respecto a los limites correspondientes a la subclase
boolean planSemanal_espacioLibre (int x)	comprueba que el plan semanal tenga espacio para aceptar trabajos con respecto a los limites correspondientes a un parametro dado
void TrabajarPruebas ()	Pone al tecnico a trabajar, procesa todas las pruebas de a una.

Methods inherited from class Trabajador

```
agregar_planSemanal, alerta_Clinica, coger_prueba, imprimir_completados,
imprimir_pacientes_Completados, imprimir_pacientes_planSemanal,
imprimir_planSemanal, in_planSemanal, iterador_planSemanal, num_casos_PS,
tiene_Trabajo
```

Methods inherited from class Person

```
contains_Tratamiento, formularioNuevoRegistro, get_apellido1, get_apellido2,
get_edad, get_identificadorDNI, get_nombre, get_tratamientos,
imprimir_Tratamientos, nueva_listaTratamientos, numero_tratamientos,
poll_Tratamiento, remove_Tratamiento, set_apellido1, set_apellido2, set_edad,
set_nombre, toString
```

Methods inherited from class java.lang.Object

```
clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait
```

Constructor Detail

Tecnico

```
public Tecnico(int DNI,
Clinica c)
```

constructor de la clase tecnico

Parameters:

DNI - el dni identificador del tecnico

c - clinica, la clinica donde va a trabajar el tecnico, necesario para las alertas

Tecnico

```
public Tecnico(Clinica c,  
               int DNI,  
               String nombre,  
               String apellido1,  
               String apellido2,  
               int edad)
```

constructor completo para las funciones de debug y pruebas. No necesario en el programa final

Method Detail

add_Tratamiento

```
public void add_Tratamiento(Tratamiento x)
```

metodo sobreescrito para agregar tratamientos a la lista con filtro. El tecnico no necesita aceptar vacunas, por que lo que el primer condicional funciona de filtro

Overrides:

`add_Tratamiento` in class `Person`

Parameters:

x - tratamiento

aceptar_Tratamiento

```
public void aceptar_Tratamiento(Tratamiento x)
```

metodo sobreescrito del metodo abstracto de la clase trabajador. Este metodo comprueba que se pueda aceptar el tratamiento, por tipo y por capacidad

Specified by:

`aceptar_Tratamiento` in class `Trabajador`

TrabajarPruebas

```
public void TrabajarPruebas()
```

Pone al tecnico a trabajar, procesa todas las pruebas de a una.

planSemanal_espacioLibre

public boolean planSemanal_espacioLibre()

comprueba que el plan semanal tenga espacio para aceptar trabajos con respecto a los limites correspondientes a la subclase

Specified by:

[planSemanal_espacioLibre](#) in class [Trabajador](#)

planSemanal_espacioLibre

public boolean planSemanal_espacioLibre(int x)

comprueba que el plan semanal tenga espacio para aceptar trabajos con respecto a los limites correspondientes a un parametro dado

Overrides:

[planSemanal_espacioLibre](#) in class [Trabajador](#)

Parameters:

x - los tratamientos que puede aceptar como limite

Returns:

puede aceptar otro tratamiento.

PACKAGE [CLASS](#) TREE INDEX HELP

ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class TestPcr

```
java.lang.Object
    Tratamiento
        Pruebas
            TestPcr
```

All Implemented Interfaces:

Serializable

```
public class TestPcr
extends Pruebas
```

Esta clase representa a un test PCR, impone un limite en los dias entre pruebas pero no tiene nada mas especial con respecto a otra prueba cualquiera de analisis de presencia viral

Version:

(0.1)

Author:

(Francisco Sanchez)

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor	Description
<code>TestPcr ()</code>	Constructor for objects of class TestPcr

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method	Description
int	<code>get_lapsoDias ()</code>	implementa el metodo abstracto de la clase padre.

Methods inherited from class Pruebas

```
analizar_prueba, get_ejecutado, get_procesado, get_resultado, hacer_prueba,
set_resultado
```

Methods inherited from class Tratamiento

```
aceptado, cuota_tiempo, get_dni_Paciente, get_Enfermero, get_fecha, get_Paciente,
get_Tecnico, get_tipoTratamiento, remove_Enfermero, remove_Paciente,
remove_Tecnico, remove_Tratamiento, set_enfermeroAsignado, set_paciente,
set_tecnico, timeStamp
```

Methods inherited from class java.lang.Object

```
clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Constructor Detail

TestPcr

```
public TestPcr()
```

Constructor for objects of class TestPcr

Method Detail

get_lapsoDias

```
public int get_lapsoDias()
```

implementa el metodo abstracto de la clase padre.

Specified by:

get_lapsoDias in class Pruebas

Returns:

lapsoDias

Class Trabajador

java.lang.Object
 Person
 Trabajador

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

Administrador, Enfermero, Tecnico

```
public abstract class Trabajador
extends Person
```

Abstract class Trabajador - write a description of the class here

Version:

(0.1)

Author:

(Francisco Sanchez)

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor	Description
Trabajador(int DNI, Clinica clinica)	

Method Summary

All Methods

Instance Methods

Abstract Methods

Concrete Methods

Modifier and Type	Method	Description
abstract void	aceptar_Tratamiento(Tratamiento t)	metodos interfaz para poder usar polimorfismo, sobreescritos en clases hijas
void	agregar_planSemanal(Pruebas x)	agrega una prueba al plan semanal

void	alerta_Clinica(Trabajador x, Tratamiento y)	envia una senhal de alerta a la clinica, comprueba por tiempo de ejecucion que tipo de trabajador esta dando la alerta y elije la senhal de acuerdo a sus limitaciones.
Pruebas	coger_prueba()	busca en la lista el elemento que necesita ser, o bien ejecutado por el enfermero o bien procesado por el tecnico y lo devuelve.
void	imprimir_completados()	imprime los tratamientos almacenados en la lista Tratamientos declarada en la clase persona.
void	imprimir_pacientes_Completados()	imprime la informacion de los pacientes de los tratamientos completados.
void	imprimir_pacientes_planSemanal()	imprime la informacion de los pacientes de los tratamientos del plan semanal
void	imprimir_planSemanal()	imprime el plan semanal
void	in_planSemanal()	mutador contructor de atributo planSemanal, representativo de la carga de trabajo correspondiente al trabajador
Iterator iterador_planSemanal()		iterador para la lista de plan semanal, permite recorrer y visualizar la lista sin alterar su composicion u orden.
int	num_casos_PS()	devuelve el tamnho de la lista del plan semanal, util para compararlo con los limites impuestos en la practica
abstract	planSemanal_espacioLibre()	
boolean		
boolean	planSemanal_espacioLibre(int limite)	devuelve la comprobacion de si el objeto trabajador tiene espacio en su lista de trabajos semanal.
boolean	tiene_Trabajo()	comprueba si la cola de trabajo semanal esta vacia

Methods inherited from class Person

add_Tratamiento, contains_Tratamiento, formularioNuevoRegistro, get_apellido1, get_apellido2, get_edad, get_identificadorDNI, get_nombre, get_tratamientos, imprimir_Tratamientos, nueva_listaTratamientos, numero_tratamientos, poll_Tratamiento, remove_Tratamiento, set_apellido1, set_apellido2, set_edad, set_nombre, toString

Methods inherited from class java.lang.Object

clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Trabajador

```
Trabajador(int DNI,  
          Clinica clinica)
```

Method Detail

aceptar_Tratamiento

```
public abstract void aceptar_Tratamiento(Tratamiento t)  
metodos interfaz para poder usar polimorfismo, sobreescritos en clases hijas
```

planSemanal_espacioLibre

```
public abstract boolean planSemanal_espacioLibre()
```

in_planSemanal

```
public void in_planSemanal()  
mutador contructor de atributo planSemanal, representativo de la carga de trabajo correspondiente al  
trabajador
```

imprimir_planSemanal

```
public void imprimir_planSemanal()  
imprime el plan semanal
```

imprimir_completados

```
public void imprimir_completados()
```

imprime los tratamientos almacenados en la lista Tratamientos declarada en la clase persona. En los pacientes representa el historial de procedimientos a los que se ha sometido, en los trabajadores los trabajos efectuados. Otro tipo de historial.

imprimir_pacientes_Completados

```
public void imprimir_pacientes_Completados()
```

imprime la informacion de los pacientes de los tratamientos completados.

imprimir_pacientes_planSemanal

```
public void imprimir_pacientes_planSemanal()
```

imprime la informacion de los pacientes de los tratamientos del plan semanal

num_casos_PS

```
public int num_casos_PS()
```

devuelve el tamnho de la lista del plan semanal, util para compararlo con los limites impuestos en la practica

iterador_planSemanal

```
public Iterator iterador_planSemanal()
```

iterador para la lista de plan semanal, permite recorrer y visualizar la lista sin alterar su composicion u orden.

planSemanal_espacioLibre

```
public boolean planSemanal_espacioLibre(int limite)
```

devuelve la comprobacion de si el objeto trabajador tiene espacio en su lista de trabajos semanal. Comprueba si alguna creada, si no la crea. Comprueba si la carga semanal es menor que el limite y devuelve que hay espacio libre en caso afirmativo. En el caso de que siga sin haber devuelto un valor true recorre la lista y comprueba los dias que han pasado entre pruebas, si hay alguna mayor de 7 dias de antiguedad taponando la lista, la elimina y la agrega a la lista de tratamientos completados.

Parameters:

limite - de tratamientos por semana

Returns:

puede aceptar otro tratamiento.

coger_prueba

```
public    Pruebas    coger_prueba()
```

busca en la lista el elemento que necesita ser, o bien ejecutado por el enfermero o bien procesado por el tecnico y lo devuelve. No lo elimina por que esto haria que la cola menguara y permitiese al trabajador aceptar mas trabajos. Por ello es requisito indispensable actualizar la cola de plan semanal y comprobar si hay elementos anteriores a 7 dias antes de meter un elemento en la cola.

tiene_Trabajo

```
public boolean tiene_Trabajo()
```

comprueba si la cola de trabajo semanal esta vacia

agregar_planSemanal

```
public void agregar_planSemanal(Pruebas    x)
```

agrega una prueba al plan semanal

Parameters:

`la` - prueba a agregar

alerta_Clinica

```
public void alerta_Clinica(Trabajador    x, Tratamiento    y)
```

envia una senhal de alerta a la clinica, comprueba por tiempo de ejecucion que tipo de trabajador esta dando la alerta y elije la senhal de acuerdo a sus limitaciones. alertaVacuna puede mandar 2 senhales, en esta version solo esta habilitada un ti

Parameters:

`e1` - trabajador que da la senhal

`la` - prueba razon de la alarma

Class Tratamiento

`java.lang.Object`
Tratamiento

All Implemented Interfaces:

`Serializable`

Direct Known Subclasses:

`Pruebas, Vacunas`

```
public abstract class Tratamiento
extends Object
implements Serializable
```

Abstract class Tratamiento - Esta clase representa de forma general cualquier procedimiento por el cual un paciente o usuario haya acudido a la clinica. Agrupa todos puesto que cualquiera, ya sea vacuna o prueba necesita almacenar una informacion muy parecida, utilizando la herencia se ahorra la duplicidad del codigo en gran medida. Tiene un alto grado de uso de polimorfismo y de comprobaciones en tiempo de ejecucion para dirigir el curso del programa

Version:

(0.1)

Author:

(Francisco Sanchez)

See Also:

`Serialized Form`

Constructor Summary

Constructors

Constructor	Description
<code>Tratamiento()</code>	constructor de la clase Tratamiento, esta es abstracta, aun asi dispone que cualquier tratamiento que se construya deba tener la fecha del momento "estampada"

Method Summary

All Methods

Instance Methods

Abstract Methods

Concrete Methods

Modifier and Type	Method	Description
-------------------	--------	-------------

boolean	<code>aceptado()</code>	metodo que comprueba si este tratamiento ha sido correctamente asignado.
boolean	<code>cuota_tiempo(int days)</code>	funcion que calcula cuantos dias han pasado desde una fecha hasta la fecha almacenada en el tratamiento.
int	<code>get_dni_Paciente()</code>	metodo selector, devuelve el identificador del paciente del tratamiento
Enfermero	<code>get_Enfermero()</code>	metodo selector, devuelve enfermero del tratamiento
LocalDate	<code>get_fecha()</code>	metodo selector que devuelve la fecha asignada al tratamiento
abstract int	<code>get_lapsoDias()</code>	metodo interfaz, permite la correcta utilizacion del polimorfismo
Paciente	<code>get_Paciente()</code>	metodo selector, devuelve paciente del tratamiento
Tecnico	<code>get_Tecnico()</code>	metodo selector, devuelve tecnico del tratamiento
String	<code>get_tipoTratamiento()</code>	devuelve una cadena con el tipo de subclase assignable al tratamiento return tipo de clase
void	<code>remove_Enfermero()</code>	metodo mutador, remueve enfermero del tratamiento
void	<code>remove_Paciente()</code>	metodo mutador, remueve paciente del tratamiento
void	<code>remove_Tecnico()</code>	metodo mutador, remueve tecnico del tratamiento
void	<code>remove_Tratamiento()</code>	metodo que se asegura de que no queda rastro del tratamiento a lo largo del programa este metodo tiene que revisarse pues contiene bugs, version actual (0.1)
void	<code>set_enfermeroAsignado(Enfermero x)</code>	metodo mutador, asigna enfermero a la prueba
void	<code>set_paciente(Paciente y)</code>	metodo mutador, asigna paciente a la prueba
void	<code>set_tecnico(Tecnico x)</code>	metodo mutador, asigna tecnico a la prueba
void	<code>timeStamp()</code>	metodo mutador que asigna la fecha actual al tratamiento

Methods inherited from class `java.lang.Object`

`clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

Tratamiento

```
public Tratamiento()
```

constructor de la clase Tratamiento, esta es abstracta, aun asi dispone que cualquier tratamiento que se construya deba tener la fecha del momento "estampada"

Method Detail

get_tipoTratamiento

```
public String get_tipoTratamiento()
```

devuelve una cadena con el tipo de subclase assignable al tratamiento return tipo de clase

set_tecnico

```
public void set_tecnico(Tecnico x)
```

metodo mutador, asigna tecnico a la prueba

aceptado

```
public boolean aceptado()
```

metodo que comprueba si este tratamiento ha sido correctamente asignado.

set_enfermeroAsignado

```
public void set_enfermeroAsignado(Enfermero x)
```

metodo mutador, asigna enfermero a la prueba

set_paciente

```
public void set_paciente(Paciente y)
```

metodo mutador, asigna paciente a la prueba

get_Paciente

public Paciente get_Paciente()

metodo selector, devuelve paciente del tratamiento

Returns:

paciente

get_dni_Paciente

public int get_dni_Paciente()

metodo selector, devuelve el identificador del paciente del tratamiento

Returns:

identificadorDNI

get_Enfermero

public Enfermero get_Enfermero()

metodo selector, devuelve enfermero del tratamiento

Returns:

enfermeroAsignado

get_Tecnico

public Tecnico get_Tecnico()

metodo selector, devuelve tecnico del tratamiento

Returns:

tecnicoAsignado

remove_Tecnico

public void remove_Tecnico()

metodo mutador, remueve tecnico del tratamiento

remove_Enfermero

public void remove_Enfermero()

metodo mutador, remueve enfermero del tratamiento

remove_Paciente

public void remove_Paciente()

metodo mutador, remueve paciente del tratamiento

remove_Tratamiento

public void remove_Tratamiento()

metodo que se asegura de que no queda rastro del tratamiento a lo largo del programa este metodo tiene que revisarse pues contiene bugs, version actual (0.1)

timeStamp

public void timeStamp()

metodo mutador que asigna la fecha actual al tratamiento

get_fecha

public LocalDate get_fecha()

metodo selector que devuelve la fecha asignada al tratamiento

cuota_tiempo

public boolean cuota_tiempo(int days)

funcion que calcula cuantos dias han pasado desde una fecha hasta la fecha almacenada en el tratamiento. Esta funcion es util para calcular los tiempos entre vacunas o tratamientos.

get_lapsoDias

```
public abstract int get_lapsoDias()  
metodo interfaz, permite la correcta utilizacion del polimorfismo
```

PACKAGE CLASS TREE INDEX HELP

ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class Vacunas

java.lang.Object
 Tratamiento
 Vacunas

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

JJ, Moderna, Pfizer

```
public abstract class Vacunas
extends Tratamiento
```

Abstract class Vacunas - Define unos patrones genericos para las 3 vacunas y los metodos aplicables como interfaz. Metodos pueden estar sobrecargados en las clases hijas para adaptarse a las necesidades propias de las subclases.

Version:

(0.1)

Author:

(Francisco Sanchez)

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor	Description
Vacunas ()	constructor de la clase Vacunas, la clase es abstracta

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type	Method	Description
boolean	cuota_tiempo ()	Comprueba, accediendo a el metodo de su clase padre si se podria realizar una segunda vacuna en el dia de la comprobacion.

int	get_dosisRestantes()	metodo selector de dosis restantes de la vacuna
int	get_lapsoDias()	devuelve el tiempo necesario de espera entre vacunas
boolean	usar()	representa la accion de aplicar una dosis de la vacuna
void	vaciar()	metodo mutador que pone a 0 la cantidad de dosis restantes en la vacuna

Methods inherited from class Tratamiento

aceptado, cuota_tiempo, get_dni_Paciente, get_Enfermero, get_fecha, get_Paciente, get_Tecnico, get_tipoTratamiento, remove_Enfermero, remove_Paciente, remove_Tecnico, remove_Tratamiento, set_enfermeroAsignado, set_paciente, set_tecnico, timeStamp

Methods inherited from class java.lang.Object

clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Vacunas

Vacunas()

constructor de la clase Vacunas, la clase es abstracta

Method Detail

usar

public boolean usar()

representa la accion de aplicar una dosis de la vacuna

Returns:

boolean si la vacuna tenia dosis suficientes devuelve true

vaciar

```
public void vaciar()  
metodo mutador que pone a 0 la cantidad de dosis restantes en la vacuna
```

get_dosisRestantes

```
public int get_dosisRestantes()  
metodo selector de dosis restantes de la vacuna
```

cuota_tiempo

```
public boolean cuota_tiempo()  
Comprueba, accediendo a el metodo de su clase padre si se podria realizar una segunda vacuna en el dia de la  
comprobacion.
```

get_lapsoDias

```
public int get_lapsoDias()  
devuelve el tiempo necesario de espera entre vacunas
```

Specified by:

`get_lapsoDias` in class `Tratamiento`

Returns:

dias

PACKAGE **CLASS** TREE INDEX HELP

ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Serialized Form

Package <Unnamed>

Class Administrador extends Trabajador implements Serializable

Class AnalisisSerologico extends Pruebas implements Serializable

Serialized Fields

lapsoDias

int lapsoDias

nivelAnticuerpos

int nivelAnticuerpos

Class Antigenos extends Pruebas implements Serializable

Serialized Fields

lapsoDias

int lapsoDias

Class BBDD extends Object implements Serializable

Serialized Fields

fichero

LinkedList<Tratamiento> fichero

neverita

Nevera neverita

numeroAdministradores

int numeroAdministradores

numeroEnfermeros

int numeroEnfermeros

numeroPacientes

int numeroPacientes

numeroTecnicos

int numeroTecnicos

registro

HashMap<Integer,Person> registro

Class *Enfermero* extends *Trabajador* implements *Serializable*

Serialized Fields

vacunas

LinkedList<Vacunas> vacunas

Class *JJ* extends *Vacunas* implements *Serializable*

Serialized Fields

lleno

boolean lleno

Class *Moderna* extends *Vacunas* implements *Serializable*

Class *Paciente* extends *Person* implements *Serializable*

Serialized Fields

primeraDosis

boolean primeraDosis

vacunado

boolean vacunado

Class *Person* extends *Object* implements *Serializable*

Serialized Fields

apellido1

String apellido1

apellido2

String apellido2

edad

int edad

identificadorDNI

int identificadorDNI

nombre

String nombre

tratamientos

LinkedList<Tratamiento> tratamientos

Class Pfizer extends Vacunas implements Serializable

Class PruebaClasica extends Antigenos implements Serializable

Class PruebaRapida extends Antigenos implements Serializable

Class Pruebas extends Tratamiento implements Serializable

Serialized Fields

ejecutado

boolean ejecutado

procesado

boolean procesado

resultado

boolean resultado

Class Tecnico extends Trabajador implements Serializable

Class *TestPcr* extends *Pruebas* implements *Serializable*

Serialized Fields

lapsoDias

int lapsoDias

Class *Trabajador* extends *Person* implements *Serializable*

Serialized Fields

clini

Clinica clini

planSemanal

LinkedList<Pruebas> planSemanal

Class *Tratamiento* extends *Object* implements *Serializable*

Serialized Fields

enfermeroAsignado

Enfermero enfermeroAsignado

fecha

LocalDate fecha

paciente

Paciente paciente

tecnicoAsignado

Tecnico tecnicoAsignado

Class Vacunas extends Tratamiento implements Serializable

Serialized Fields

dosisRestantes

int dosisRestantes

[PACKAGE](#) [CLASS](#) [TREE](#) [INDEX](#) [HELP](#)

[ALL CLASSES](#)

Constant Field Values

Contents

<Unnamed>

<Unnamed>.*

Antigenos

Modifier and Type	Constant Field	Value
final int	lapsoDias	1

estatico

Modifier and Type	Constant Field	Value
static final int	ENF_LIMITE	5
static final int	NUSUARIOTEST	100
static final int	PCR	15
static final int	PROBABILIDADNEGATIVO	7
static final int	SEROLOGICO	180
static final int	TEC_LIMITE	4
static final int	TIEMPOVACUNAS	21

All Classes

Class Summary

Class	Description
Administrador	Esta clase representa al administrador de la clinica, las funciones del administrador estan declaradas en la clinica y esta comprueba que el usuario es un administrador para permitir la manipulacion de esta.
AnalisisSerologico	Analisis serologico hereda de prueba aunque es un poco mas especial que el resto, esta prueba no determina si se tiene el virus, si no si se hayan presentes en el organismo los anticuerpos para este.
Antigenos	Abstract class Antigenos - clase abstracta, representa el comun de las pruebas de antigenos, las cuales se pueden hacer diariamente.
BBDD	La base de datos de la que se nutre la clase clinica y donde se almacena toda la informacion, implementa la interfaz Serializable para poder hacer una copia y mandarlo como stream de digitos binarios a un archivo fuera del entorno de bluej y a la inversa, para cargar un objeto base de datos de un archivo independiente y restaurar la informacion de una ejecucion anterior.
Clinica	Clase principal del programa, desde aqui se administra y se consultan los estados de este.
Enfermero	
estatico	Clase que almacena funciones, metodos y parametros de utilidad general en el programa.
JJ	Write a description of class JJ here.
Moderna	Write a description of class Moderna here.
ModuloConfinamiento	Esta clase se encarga de los confinamientos, utiliza una clase privada NodoConfinamiento, compuesta de 3 listas las cuales van pasandose las pruebas segun se confirme que el paciente esta aislando.
ModuloVacunacion	Write a description of class ModuloVacunacion here.
Nevera	Clase que representa una nevera donde se almacenan las vacunas
Paciente	Paciente es una subclase de Person, se construye con un digito dni que representa un valor unico, consta de dos atributos extras con respecto a la clase persona, primeraDosis que representa la situacion de que el paciente haya recibido una dosis de dos y vacunado, que representa la totalidad de la vacuacion.
Person	Abstract class Person - Superclase, define los atributos de un objeto abstracto persona Atributos principales designados en la clase: Identificador Nombre Apellido Apellido Edad lista de tratamientos (el significado de esta lista varia en funcion de la subclase)

Pfizer	Write a description of class Pfizer here.
PruebaClasica	Esta clase solo diferencia entre tipos de pruebas de antigenos y no contiene ninguna diferencia con respecto a la clase padre
PruebaRapida	Esta clase solo diferencia entre tipos de pruebas de antigenos y no contiene ninguna diferencia con respecto a la clase padre
Pruebas	Abstract class Pruebas - clase de la que derivan las pruebas, esta a su vez hereda de tratamientos consta de 3 atributos mas con respecto a Tratamientos; resultado, procesado y ejecutado.
Tecnico	Funciones del tecnico: -Visualizacion de datos de los pacientes asignados -Registro y actualizacion de pruebas diagnosticas y vacunacion
TestPcr	Esta clase representa a un test PCR, impone un limite en los dias entre pruebas pero no tiene nada mas especial con respecto a otra prueba cualquiera de analisis de presencia viral
Trabajador	Abstract class Trabajador - write a description of the class here
Tratamiento	Abstract class Tratamiento - Esta clase representa de forma general cualquier procedimiento por el cual un paciente o usuario haya acudido a la clinica.
Vacunas	Abstract class Vacunas - Define unos patrones genericos para las 3 vacunas y los metodos aplicables como interfaz.

[PACKAGE](#) [CLASS](#) [TREE](#) [INDEX](#) [HELP](#)

[ALL CLASSES](#)

PACKAGE CLASS TREE INDEX HELP

ALL CLASSES

SEARCH: 

All Packages

Package Summary

Package	Description
<Unnamed>	

PACKAGE CLASS TREE INDEX HELP

ALL CLASSES

Hierarchy For All Packages

Class Hierarchy

- [java.lang.Object](#)
 - [BBDD](#) (implements `java.io.Serializable`)
 - [Clinica](#)
 - [estatico](#)
 - [ModuloConfinamiento](#)
 - [ModuloVacunacion](#)
 - [Nevera](#)
 - [Person](#) (implements `java.io.Serializable`)
 - [Paciente](#)
 - [Trabajador](#)
 - [Administrador](#)
 - [Enfermero](#)
 - [Tecnico](#)
- [Tratamiento](#) (implements `java.io.Serializable`)
 - [Pruebas](#)
 - [AnalisisSerologico](#)
 - [Antigenos](#)
 - [PruebaClasica](#)
 - [PruebaRapida](#)
 - [TestPcr](#)
 - [Vacunas](#)
 - [JJ](#)
 - [Moderna](#)
 - [Pfizer](#)

```
1 /**
2  * Write a description of class JJ here.
3  *
4  * @author (Francisco Sanchez)
5  * @version (0.1)
6  */
7
8 public class JJ extends Vacunas
9 {
10
11     boolean lleno;
12
13     /**
14      * Constructor for objects of class JJ
15      */
16     public JJ( )
17     {
18         super();
19         lleno = true;
20     }
21
22
23     @Override
24     public boolean usar()
25     {
26
27         if(lleno) {lleno = false; this.vaciar(); return true; }
28         else      return false;
29     }
30
31 }
32 }
33 }
```

```
1 /**
2  * Abstract class Vacunas - Define unos patrones genericos para las 3 vacunas
3  * y los metodos aplicables como interfaz. Metodos pueden estar sobrecargados en las clases
4  * hijas para adaptarse a las necesidades propias de las subclases.
5  *
6  * @author (Francisco Sanchez)
7  * @version (0.1)
8  */
9
10 public abstract class Vacunas extends Tratamiento
11 {
12
13
14
15     private int dosisRestantes;
16
17     /**
18      * constructor de la clase Vacunas, la clase es abstracta
19      */
20     Vacunas()
21     {
22
23         dosisRestantes = 2; //representa una dosis total, en jj al usar la vacuna se restan las dos a la vez
24     }
25
26     /**
27      * representa la accion de aplicar una dosis de la vacuna
28      * @return boolean si la vacuna tenia dosis suficientes devuelve true
29      */
30     public boolean usar()
31     {
32         if(dosisRestantes >= 1)
33         {
34             dosisRestantes--;
35             return true;
36         }
37         else
38         {
39             return false;
40         }
41     }
42
43     /**
44      * metodo mutador que pone a 0 la cantidad de dosis restantes
45      * en la vacuna
46      */
47     public void vaciar()
48     {
49         this.dosisRestantes = 0;
50     }
51
52     /**
53      * metodo selector de dosis restantes de la vacuna
54      */
55     public int get_dosisRestantes()
56     {
57         return this.dosisRestantes;
58     }
59
60     /**
61      * Comprueba, accediendo a el metodo de su clase padre si se podria
62      * realizar una segunda vacuna en el dia de la comprobacion.
63      *
64      */
65     public boolean cuota_tiempo()
66     {
67         return super.cuota_tiempo(estatico.TIEMPOVACUNAS);
68     }
69
70     /**
71      * devuelve el tiempo necesario de espera entre vacunas
72      * @return dias
73      */
74     public int get_lapsoDias()
75     {
76         return estatico.TIEMPOVACUNAS;
77     }
78 }
```

```
1 /**
2  * Analisis serologico hereda de prueba aunque es un poco mas especial que el
3  * resto, esta prueba no determina si se tiene el virus, si no si se hayan
4  * presentes en el organismo los anticuerpos para este.
5  * Por lo tanto el resultado determina en este caso si se esta "inmunizado".
6  * Hay que prestar especial atencion para no mezclarlo con los otros.
7  *
8  */
9  * @author (Francisco Sanchez)
10 * @version (0.1)
11 */
12 public class AnalisisSerologico extends Pruebas
13 {
14
15     private int nivelAnticuerpos;
16     private final int lapsoDias = estatico.SEROLOGICO;
17
18     /**
19      * constructor del analisis serologico. el constructor llama a super() automaticamente.
20      */
21     public AnalisisSerologico()
22     {
23         nivelAnticuerpos = 0;
24     }
25
26
27
28     /**
29      * metodo que sobreescribe al metodo de la clase padre analizar_prueba.
30      * La clase difiere del resto y por lo tanto necesita un trato especial.
31      * @param el tecnico que va a analizar la prueba
32      */
33     @Override
34     public void analizar_prueba(Tecnico tecnico)
35     {
36         this.nivelAnticuerpos = estatico.resultadoAnticuerpos();
37         this.set_resultado(nivelAnticuerpos > 2);
38     }
39
40     /**
41      * metodo accesror del tiempo en dias entre pruebas de tipo serologico.
42      * @return estatico.SEROLOGICO
43      */
44     @Override
45     public int get_lapsoDias()
46     {
47         return this.lapsoDias;
48     }
49 }
50 }
```

```
1 /**
2  * Abstract class Antigenos -
3  * clase abstracta, representa el comun de las pruebas
4  * de antigenos, las cuales se pueden hacer diariamente.
5  *
6  * @author (Francisco Sanchez)
7  * @version (0.1)
8  */
9
10 public abstract class Antigenos extends Pruebas
11 {
12
13     final int lapsoDias = 1;
14
15
16
17
18     /**
19      * implementa el metodo abstracto de la clase padre.
20      * @return lapsoDias
21      */
22     @Override
23     public int get_lapsoDias()
24     {
25         return this.lapsoDias;
26     }
27
28
29
30
31 }
32 }
```

```
1 /**
2  * Write a description of class ModuloVacunacion here.
3  *
4  * @author (your name)
5  * @version (a version number or a date)
6  */
7
8 public class ModuloVacunacion
9 {
10    // instance variables - replace the example below with your own
11    private int x;
12
13    /**
14     * Constructor for objects of class ModuloVacunacion
15     */
16    public ModuloVacunacion()
17    {
18        // initialise instance variables
19        x = 0;
20    }
21
22    /**
23     * An example of a method - replace this comment with your own
24     *
25     * @param y a sample parameter for a method
26     * @return the sum of x and y
27     */
28    public int sampleMethod(int y)
29    {
30        // put your code here
31        return x + y;
32    }
33}
34
```

```
1 import java.util.LinkedList;
2 import java.time.LocalDate;
3 import java.time.temporal.ChronoUnit;
4
5 /**
6  * Paciente es una subclase de Person, se construye con un digito dni que representa un valor unico,
7  * consta de dos atributos extras con respecto a la clase persona, primeraDosis que representa la situacion
8  * de que el paciente haya recibido una dosis de dos y vacunado, que representa la totalidad de la vacuacion.
9  * Si se administra una vacuna de 1 sola dosis se validan los dos. No es necesaria la diferenciacion
10 */
11
12 */
13 * @author (Francisco Sanchez)
14 * @version (0.1)
15 */
16 public class Paciente extends Person
17 {
18
19
20     private boolean primeraDosis;
21     private boolean vacunado;
22
23     /**
24      * Constructor de la clase paciente
25      */
26     public Paciente(int DNI)
27     {
28         super(DNI);
29         nueva_listaTratamientos();
30         vacunado = false;
31         primeraDosis = false;
32     }
33
34     /**
35      * Constructor de la clase paciente con atributos directos, no necesario en el
36      * programa final pero util para hacer pruebas.
37      */
38     public Paciente( int DNI, String nombre, String apellido1, String apellido2, int edad)
39     {
40         super(DNI);
41         nueva_listaTratamientos();
42         vacunado = false;
43         primeraDosis = false;
44         this.set_edad(edad);
45         this.set_nombre(nombre);
46         this.set_apellido1(apellido1);
47         this.set_apellido2(apellido2);
48     }
49
50     /**
51      * Devuelve una cadena de caracteres con el nombre y apellido de la persona representada en el
52      * objeto
53      *
54      * @return String nombre y apellido del paciente
55      */
56     public String nomYapell()
57     {
58         return (this.get_nombre() + " " + this.get_apellido1());
59     }
60
61     /**
62      * Administra una dosis de vacuna al objeto paciente, altera el estado del objeto
63      * vacuna y lo anhade a la cola de tratamientos si es la primera vez que se usa en
64      * el paciente.
65      */
66     public void vacunar(Vacunas x)
67     {
68         if(x instanceof JJ){
69             this.primeraDosis = true;
70             this.vacunado = x.usar();
71             this.add_Tratamiento(x);
72         }else{
73             if(this.primeraDosis = false)
74             {
75                 this.primeraDosis = x.usar();
76                 this.add_Tratamiento(x);
77             }else
78             {
```

```
79         vacunado = x.usar();
80     }
81 }
82 System.out.println("Paciente: \"auch!\"");
83 }
84
85 /**
86 * confirma que haya recibido su primera dosis de vacuna
87 * @return primeraDosis ha recibido primera dosis
88 */
89 public boolean get_estadoPrimeraDosis(){
90     return this.primeraDosis;
91 }
92
93 /**
94 * confirma que haya recibido la totalidad de la vacunacion
95 * @return vacunado el estado de vacunacion del usuario
96 */
97     public boolean get_estadoVacunacion()
98 {
99     return this.vacunado;
100 }
101
102 /**
103 * Comprueba que el paciente pueda someterse a un analisis serologico
104 * @return boolean
105 */
106 public boolean puede_AnalisisSeroLogico()
107 {
108     return puede_entrePruebas(1);
109 }
110
111 /**
112 * Comprueba que el paciente pueda someterse a un test PCR
113 * @return boolean
114 */
115 public boolean puede_TestPcr()
116 {
117     return puede_entrePruebas(2);
118 }
119
120 /**
121 * Comprueba que el paciente pueda someterse a una vacunacion;
122 * primera o segunda dosis.
123 * @return boolean
124 */
125 public boolean puede_Vacuna()
126 {
127     if(!primeraDosis && !vacunado)
128     { return true;}
129     else{
130         return puede_entrePruebas(3);
131     }
132 }
133
134 /**
135 * El metodo ejecutor de las comprobaciones para las pruebas,
136 * hace uso del polimorfismo para evitar errores en tiempo de
137 * ejecucion. Comprueba el tiempo habilitante entre pruebas de
138 * una variable estatica en una clase concreta, definidora de
139 * parametros.
140 *
141 * imprime la informacion relativa a los dias restantes para la prueba
142 * en caso de que no se pueda repetir en el mismo dia
143 */
144 public boolean puede_entrePruebas(int seleccion)
145 {
146     boolean puede = true;
147     LocalDate aux = LocalDate.now();
148     LinkedList<? extends Tratamiento> lista = get_tratamientos();
149     int dias = 0;
150     Tratamiento conmutador = null;
151     switch(seleccion)
152     {
153     case 1:
154         System.out.println("SeroLogico");
155         System.out.println(lista.size() );
156         for(Tratamiento x : lista)
```

```
157 {
158     if( x instanceof AnalisisSerologico)
159     {
160
161         System.out.println(x.get_fecha());
162
163         dias = (int)ChronoUnit.DAYS.between(x.get_fecha(), aux);
164         System.out.println("dias entre fechas" + dias);
165         dias = dias - estatico.SEROLOGICO;
166         System.out.println("- serologico" + dias);
167         if(dias < 7) {
168             puede = false;
169         }
170     }//else {System.out.println("No lo reconoce");}
171 }
172
173
174     break;
175 case 2:
176     System.out.println("PCR");
177     for(Tratamiento x : lista)
178     {
179         if( x instanceof TestPcr)
180         {
181             dias = (int)ChronoUnit.DAYS.between(x.get_fecha(), aux);
182             dias = dias - estatico.PCR;
183             if(dias < 7) {
184                 puede = false;
185             }else{
186                 puede = true;
187             }
188         }
189     }
190
191
192     break;
193
194 case 3:
195     System.out.println("vacuna");
196     for(Tratamiento x : lista)
197     {
198         if( x instanceof Vacunas)
199         {
200             dias = (int)ChronoUnit.DAYS.between(x.get_fecha(), aux);
201             dias = dias - estatico.TIEMPOVACUNAS;
202             if(dias < 7) {
203                 puede = false;
204             }else{
205                 puede = true;
206             }
207         }
208     }
209 default:
210     System.out.println("Default");
211     return true;
212
213 }
214
215 return puede;
216
217 }
218
219
220
221 }
222
223 }
```

```
1 import java.io.BufferedReader;
2 import java.io.InputStreamReader;
3 import java.util.Set;
4 import java.util.LinkedList;
5
6 /**
7 * Clase principal del programa, desde aqui se administra y se
8 * consultan los estados de este.
9 */
10 * @author (Francisco)
11 * @version (0.1)
12 */
13 public class Clinica
14 {
15     private BBDD baseDatos;
16     private Person usuario;
17     private ModuloConfinamiento confinamiento;
18     int umbral;
19
20 /**
21 * Constructor for objects of class Clinica
22 */
23 public Clinica()
24 {
25     baseDatos = null;
26     usuario = null;
27     confinamiento = new ModuloConfinamiento(this);
28     umbral = 1;
29 }
30
31 /**
32 * Metodo principal de la clase, todas las funciones se controlan desde aqui
33 */
34
35 public void iniciar()
36 {
37
38     if(baseDatos == null)
39     {
40         System.out.println("Quiere cargar una base de datos(1), \n      crear una nueva?(2)");
41         switch(estatico.pedirNumero(1,2))
42         {
43             case 1:
44                 System.out.println("Modulo a implementar en version 0.2 implementando serializable");
45
46             case 2:
47                 cargaBBDD();
48                 break;
49
50         }
51
52     }
53
54
55
56     }
57
58     cargaUsuario();
59     menuUsuarios();
60 }
61
62 /**
63 * Por implementar, carga base de datos desde objeto serializado en archivo
64 */
65 private void cargaBBDD()
66 {
67
68     System.out.println("No hay una base de datos cargada, creando...");
69     baseDatos = new BBDD();
70
71 }
72
73
74
75 /**
76 * Controlador de la clinica, carga el usuario y el menu correspondiente a su categoria
77 */
78 private void cargaUsuario()
```

```
79    {
80
81        String entrada;
82        int entradaInt;
83        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
84        boolean error = false;
85
86
87
88        if(baseDatos.size_registro() == 0)
89        {
90            System.out.println("Creando administrador;\n Introduzca DNI de administrador");
91            entradaInt = estatico.pedirNumero();
92            Administrador administrator = new Administrador(entradaInt, this);
93            administrator.formularioNuevoRegistro();
94            baseDatos.add_registro(entradaInt, administrator);
95            usuario = administrator;
96        }  
else
97        {
98            System.out.println("Por favor, introduzca su DNI");
99            entradaInt = estatico.pedirNumero();
100           if(baseDatos.contains_registro(entradaInt))
101           {
102               usuario = baseDatos.get_registro(entradaInt);
103           }  
else{
104               System.out.println(" no se ha encontrado el usuario, por favor"
105                         + "\n contacte con su administrador.");
106           }
107       }
108   }
109
110 /**
111 * template de menus
112 */
113 private void menuUsuarios()
114 {
115     if(usuario instanceof Administrador)
116     {
117         menuAdministrador((Administrador)this.usuario);
118     }  
else{
119     if(usuario instanceof Trabajador)
120     {
121         menuTrabajador((Trabajador)this.usuario);
122     }  
else{
123     if(usuario instanceof Paciente)
124     {
125         System.out.println("Usted es un paciente de esta clinica, si necesita ayuda");
126         System.out.println("por favor, contacte con un trabajador");
127     }
128     if(usuario == null)
129     {
130         System.out.println("no se ha encontrado el usuario");
131     }
132   }
133 }
134 }
135
136 /**
137 * Menu especifico de administrador
138 */
139 private void menuAdministrador(Administrador x)
140 {
141     boolean salir = false;
142     int entrada;
143
144     do{
145         System.out.println("      Bienvenido Administrador ");
146         System.out.println("seleccione la operacion que desea ejecutar:");
147         System.out.println("      * 1 Gestion de usuarios ");
148         System.out.println("      * 2 Asignacion de tratamientos (pruebas y vacunas)");
149         System.out.println("      * 3 Visualizacion de datos ");
150         System.out.println("      * 4 Modulo Confinamiento");
151         System.out.println("      * 5 Actualizacion del stock de vacunas");
152         System.out.println("      * 6 Visualizacion planificacion de vacunas");
153         System.out.println("      * 7 Cerrar sesion");
154
155
156     entrada = estatico.pedirNumero(1,8);
```



```
234     \n");
235     System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n");
236     System.out.println("Se ha borrado la base de datos, su usuario administrador solo");
237     System.out.println("existe en la sesion actual, ");
238     System.out.println("dni del programador, 75910907 <- copie y pegue");
239     System.out.println("\nXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
240     System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
241   }
242   System.out.println("RECUERDE, DESDE MENU ADMINISTRADOR, OPCION 8 MODO DEBUG");
243 }
244
245 /**
246 * Menu que permite la modificacion o eliminacion de usuarios del sistema
247 * @param int dni de usuario, pedido en metodo.
248 */
249
250 private void gestionUsuarios(Administrador t)
251 {
252   int seleccion;
253   System.out.println("\n\n Administracion de usuarios");
254   System.out.println("Introduzca DNI ");
255   seleccion = estatico.pedirNumero();
256   Person auxiliar = null;
257   if(baseDatos.contains_registro(seleccion)){
258     System.out.println("se ha encontrado usuario");
259     System.out.println("\n * 1 Modificar usuario");
260     System.out.println(" * 2 Eliminar usuario");
261     System.out.println(" * 0 Salir");
262
263     switch(estatico.pedirNumero(0,2))
264     {
265       case 1:
266         auxiliar = baseDatos.get_registro(seleccion);
267         auxiliar.formularioNuevoRegistro();
268         break;
269       case 2:
270         baseDatos.eliminar_registro(seleccion);
271         break;
272       case 0:
273         break;
274     }
275   }else
276   {
277     System.out.println("No se ha encontrado al usuario");
278     this.altaUsuario((Administrador)this.usuario, seleccion);
279   }
280 }
281
282 /**
283 * Metodo que crea usuarios por eleccion
284 */
285
286 public void altaUsuario(Administrador t, int DNI)
287 {
288   Person auxiliar = null;
289   System.out.println("Desea dar de alta a un nuevo usuario? (S)i");
290   if(estatico.pedirConfirmacion())
291   {
292     System.out.println("Seleccione categoria");
293     System.out.println("\n * 1 Paciente");
294     System.out.println(" * 2 Tecnico");
295     System.out.println(" * 3 Enfermero");
296     System.out.println(" * 4 Administrador");
297     switch(estatico.pedirNumero(1,4))
298     {
299       case 1:
300         auxiliar = new Paciente(DNI);
301         break;
302       case 2:
303         auxiliar = new Tecnico(DNI, this);
304         break;
305       case 3:
306         auxiliar = new Enfermero(DNI, this);
307         break;
308       case 4:
309         auxiliar = new Administrador(DNI, this);
310         break;
```

```
312     if(auxiliar !=null){
313         auxiliar.formularioNuevoRegistro();
314         baseDatos.add_registro(DNI, auxiliar);
315     }
316 }
317 }
318 }
319 }
320 /**
321 * metodo que asigna pruebas a los diferentes diferentes usuarios,
322 * @param tipo tipo de prueba, enumerada: 1 Serologico 2 PCR 4 Vacuna y 3 Antigeno
323 * @param paciente usuario al que asignar la prueba
324 * @param usuario de la clinica, medida de seguridad impuesta en tiempo de ejecucion,
325 * si la clase dada por parametro no es un admin e intenta acceder falla el programa.
326 */
327 public void asignar_prueba(int tipo, Paciente x, Administrador y)
328 {
329     // 1 serologico 2 PCR 3 Vacuna 4 antigenos (1)dia
330     Tratamiento tratamiento = null;
331     LinkedList<Trabajador> lista = null;
332
333     if(x.puede_entrePruebas(tipo))
334     {
335         switch(tipo)
336         {
337             case 1:
338                 tratamiento = new AnalisisSeroLogico();
339                 break;
340             case 2:
341                 tratamiento = new TestPcr();
342                 break;
343             case 4:
344                 tratamiento = baseDatos.get_vacuna_Random();
345                 System.out.println("Vacuna??? Clinica\\asignar_prueba");
346                 break;
347             case 3:
348                 System.out.println("Clasica(1) o Rapida(2)");
349                 switch(estatico.pedirNumero(1,2))
350                 {
351                     case 1:
352                         tratamiento = new PruebaClasica();
353                         break;
354                     case 2:
355                         tratamiento = new PruebaRapida();
356                         break;
357                 }
358                 break;
359             default:
360                 System.out.println("fallido al introducir numero Clinica\\asignar_prueba");
361                 break;
362         }
363
364         if(!(tratamiento instanceof Vacunas) && (trabajadoresLibres()))
365         {
366             baseDatos.add_Tratamiento(tratamiento);
367             x.add_Tratamiento(tratamiento);
368         }
369
370         tratamiento.set_paciente(x);
371
372         if(tratamiento != null && !(tratamiento instanceof Vacunas)
373             &&(trabajadoresLibres()))
374         {
375             lista = baseDatos.lista_Tecnicos();
376             try
377             {
378                 for(Trabajador z : lista)
379                 {
380                     if(z.planSemanal_espacioLibre())
381                     {
382                         tratamiento.set_tecnico((Tecnico)z);
383                         z.aceptar_Tratamiento(tratamiento);
384                         throw new RuntimeException();
385                     }
386                 }
387             }catch(Exception e)
388             {
389                 System.out.println("Tecnico: Aceptado");
```

```
390     }
391
392
393     lista = baseDatos.lista_Enfermero();
394     try
395     {
396         for(Trabajador g : lista)
397         {
398             if(g.planSemanal_espacioLibre())
399             {
400                 tratamiento.set_enfermeroAsignado((Enfermero)g);
401                 g.aceptar_Tratamiento(tratamiento);
402                 throw new RuntimeException();
403             }
404         }
405     }catch(Exception e)
406     {
407         System.out.println("Enfermero: Aceptado");
408     }
409 } else if (tratamiento != null){
410     boolean aceptado = false;
411
412     lista = baseDatos.lista_Enfermero();
413
414     while(aceptado = false){
415
416         for(Trabajador g : lista)
417         {
418             try{
419                 if (((Enfermero)g).vacunacionesRestantes() < umbral)
420                 {
421                     g.aceptar_Tratamiento(tratamiento);
422                     aceptado = true;
423                     throw new RuntimeException();
424                 }
425             }catch(Exception e)
426             {
427                 System.out.println("Enfermero acepto vacuna, confirmar"
428                     +"con modulo vacunacion");
429             }
430         }
431     if(aceptado = false) umbral++;
432
433     }
434
435
436
437     }else{System.out.println("no hay suficientes trabajadores libres");}
438 }else{
439     System.out.println("No ha pasado el tiempo suficiente entre pruebas");
440 }
441
442
443 }
444
445
446 /**
447 * metodo debug
448 * @param tipo tipo de prueba, enumerada: 1 Serologico 2 PCR 4 Vacuna y 3 Antigeno
449 * @param paciente usuario al que asignar la prueba
450 * @param usuario de la clinica, medida de seguridad impuesta en tiempo de ejecucion,
451 * si la clase dada por parametro no es un admin e intenta acceder falla el programa.
452 */
453 public void asignar_prueba(int tipo, Paciente x, Administrador y, Tecnico z, Enfermero l)
454 {
455     // 1 serologico 2 PCR 3 Vacuna 4 antigenos (1)dia
456     Tratamiento tratamiento = null;
457
458
459     if(x.puede_entrePruebas(tipo))
460     {
461         switch(tipo)
462         {
463             case 1:
464                 tratamiento = new AnalisisSerologico();
465                 break;
466             case 2:
467                 tratamiento = new TestPcr();
```

```
468         break;
469     case 3:
470         tratamiento = baseDatos.get_vacuna_Random();
471         System.out.println("Vacuna?? Clinica\\asignar_prueba");
472         break;
473     case 5:
474
475         tratamiento = new PruebaClasica();
476         break;
477     case 6:
478         tratamiento = new PruebaRapida();
479         break;
480     }
481
482
483
484     if(!(tratamiento instanceof Vacunas) && (trabajadoresLibres()))
485     {
486         baseDatos.add_Tratamiento(tratamiento);
487         x.add_Tratamiento(tratamiento);
488     }
489
490     tratamiento.set_paciente(x);
491
492     if(tratamiento != null && !(tratamiento instanceof Vacunas)
493     &&(trabajadoresLibres()))
494     {
495         if(z.planSemanal_espacioLibre())
496         {
497             tratamiento.set_tecnico((Tecnico)z);
498             z.aceptar_Tratamiento(tratamiento);
499
500             System.out.println("Tecnico: Aceptado");
501         }
502
503
504
505         if(l.planSemanal_espacioLibre())
506         {
507             tratamiento.set_enfermeroAsignado((Enfermero)l);
508             l.aceptar_Tratamiento(tratamiento);
509         }
510
511
512
513     } else if (tratamiento != null)
514     {
515         l.aceptar_Tratamiento(tratamiento);
516
517     }
518
519
520
521     } System.out.println("No ha pasado el tiempo suficiente entre pruebas");
522 }
523
524 /**
525 * Metodo decorador del metodo asignar_prueba(), hace las veces de comprobador de estado de la clinica
526 * y configurador de parametros, tiene algunos metodos bloqueados a la espera de ser implementados (version
0.1)
527 */
528 public void asignarTratamiento(Administrador t)
529 {
530     int apoyo;
531     Tratamiento auxiliar;
532     Person complemento;
533
534     System.out.println(" Menu Asignacion pruebas y vacuNas (Sistema MAN)");
535     System.out.println("Por favor, ingrese el DNI del sujeto a MANipular");
536     apoyo = estatico.pedirNumero();
537
538     if(baseDatos.contains_registro(apoyo))
539     {
540         complemento = baseDatos.get_registro(apoyo);
541         System.out.println("\nPor favor, elija el tratamiento:");
542         System.out.println( "      * 1 Vacuna");
543         System.out.println( "      * 2 Pruebas");
544
545         switch(estatico.pedirNumero(1,2))
```

```
546 |
547 |     {
548 |         case 1:
549 |             // System.out.println("\n Desea que el proceso sea automatico(1) o manual?(2)");
550 |             // switch(estatico.pedirNumero(1,2))
551 |             switch(2)
552 |             {
553 |                 case 1:
554 |                     //asignar_vacunaAutomatica();
555 |                     break;
556 |                 case 2:
557 |                     //asignar_vacunaManual();
558 |                     break;
559 |             }
560 |             break;
561 |         case 2:
562 |             System.out.println("Elija el tipo de prueba");
563 |             System.out.println("    * 1 Analisis Serologico");
564 |             System.out.println("    * 2 Test PCR");
565 |             System.out.println("    * 3 Antigenos:");
566 |             System.out.println("          * Clasica");
567 |             System.out.println("          * Prueba rapida");
568 |             if(complemento instanceof Paciente )
569 |             {
570 |                 if(trabajadoresLibres())
571 |                     asignar_prueba(estatico.pedirNumero(1,4), ((Paciente)complemento), t);
572 |                 else System.out.println("No hay trabajadores libres, contrate mas");
573 |             }else{
574 |                 System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
575 |                 System.out.println("Ha elegido un usuario no paciente, si desea hacerle");
576 |                 System.out.println("una prueba a un trabajador debe crearle una ficha como");
577 |                 System.out.println("paciente, actualmente no esta implementado pero puede usted");
578 |                 System.out.println("acceder a esta funcion creando un usuario con el mismo dni");
579 |                 System.out.println(" + un numero identificativo interno, como la edad, por ejemplo");
580 |                 System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
581 |             }
582 |             break;
583 |         }
584 |
585 |         System.out.println("");
586 |     }
587 |
588 | }
589 |
590 | /**
591 | * metodo que comprueba que haya trabajadores con espacio en su plan semanal
592 | * @return boolean
593 | */
594 |
595 | public boolean trabajadoresLibres()
596 | {
597 |     boolean tecnicoLibre = false;
598 |     boolean enfermeroLibre = false;
599 |
600 |     LinkedList<Trabajador> lista = baseDatos.lista_Tecnicos();
601 |     try
602 |     {
603 |         for(Trabajador g : lista)
604 |         {
605 |             if(g.planSemanal_espacioLibre())
606 |             {
607 |                 tecnicoLibre = true;
608 |                 throw new RuntimeException();
609 |             }
610 |         }
611 |     }catch(Exception e){}
612 |
613 |     lista = baseDatos.lista_Enfermero();
614 |     try
615 |     {
616 |         for(Trabajador g : lista)
617 |         {
618 |             if(g.planSemanal_espacioLibre())
619 |             {
620 |                 enfermeroLibre = true;
621 |                 throw new RuntimeException();
622 |             }
623 |         }
624 |     }
```

```
624         }catch(Exception e){}
625
626         return (tecnicoLibre && enfermeroLibre);
627     }
628 }
629
630 /**
631 * metodo que asigna un tratamiento de forma directa, sin pedir parametros al usuario.
632 * Implementado como una forma de acceder a las clases para el modo debug y pruebas.
633 * Este metodo hace un bypass a los comrpobadores de estados presentes en la cadena
634 * de sucesion de metodos presente en el programa principal
635 *
636 * No necesario en el programa final.
637 *
638 * @param Administrador , restringe el uso del metodo a los administradores
639 * @param DNIpaciente , el identificador del paciente que se va a someter a las pruebas
640 * @param tipo de tratamiento a crear y asignar.
641 */
642
643 public void asignarTratamiento(Administrador t, int DNIpaciente, int tipo)
644 {
645     int apoyo;
646     Tratamiento auxiliar;
647     Person complemento;
648     if(baseDatos.contains_registro(DNIpaciente))
649     {
650         complemento = baseDatos.get_registro(DNIpaciente);
651
652         //"
653         // "
654         // "
655         // "
656         // "
657         if(complemento instanceof Paciente)
658         {
659             asignar_prueba(tipo, ((Paciente)complemento), t);
660         }else{
661             System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
662             System.out.println("Ha elegido un usuario no paciente, si desea hacerle");
663             System.out.println("una prueba a un trabajador debe crearle una ficha como");
664             System.out.println("paciente, actualmente no esta implementado pero puede usted");
665             System.out.println("acceder a esta funcion creando un usuario con el mismo dni");
666             System.out.println(" + un numero identificativo interno, como la edad, por ejemplo");
667             System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
668         }
669     }
670 }
671
672
673     System.out.println("");
674 }
675
676
677
678 /**
679 * String de apoyo, otorga claridad al metodo imprimirDatos, el String esta formateado.
680 */
681
682 private String imp()
683 {
684     return (String.format("%-15s", "Nombre") + String.format("%-15s", "Apellido")
685     + String.format("%-15s", "Apellido") + String.format("%-5s", "edad")
686     + String.format("%-8s", "DNI"));
687 }
688
689 /**
690 * Imprime el estado general de la clinica
691 */
692 private void imprimirDatos()
693 {
694     Person auxiliar;
695     Set<Integer> recorrido = baseDatos.get_archivo().keySet();
696
697     System.out.println("*****");
698     System.out.println("\n\nLa clinica tiene un registro de personas total de "
699     + baseDatos.size_registro());
700     System.out.println();
```

```
701
702     System.out.println("Datos de Pacientes: ");
703     System.out.println("-----");
704     System.out.print(imp() + "    vacunado   ");
705     System.out.println();
706     for(int x: recorrido)
707     {
708
709         auxiliar = baseDatos.get_registro(x);
710         if(auxiliar instanceof Paciente){
711             Paciente aux = (Paciente)auxiliar;
712             System.out.println(aux.toString()+"    "+String.format("%-8s",aux.get_estadoVacunacion()));
713             if(((Paciente)auxiliar).get_estadoPrimeraDosis()) System.out.print(" 1 dosis recibida");
714
715         }
716     }
717
718     System.out.println();
719     System.out.println("Datos de Tecnicos: ");
720     System.out.println("-----");
721     System.out.print(imp() + "no. casos: cc plan semanal");
722     System.out.println();
723     for(int x : recorrido)
724     {
725
726         auxiliar = baseDatos.get_registro(x);
727
728         if(auxiliar instanceof Tecnico){
729             Tecnico aux = (Tecnico)auxiliar;
730
731             System.out.println(aux.toString() + "      "+String.format("%-15d",aux.numero_tratamientos())
732             + String.format("%-4d",((Trabajador)auxiliar).num_casos_PS()));
733         }
734
735     }
736
737     System.out.println();
738     System.out.println("Datos de Enfermeros:");
739     System.out.println("-----");
740     System.out.print(imp() + "no. casos: cc plan semanal");
741     System.out.println();
742     for(int x : recorrido)
743     {
744
745         auxiliar = baseDatos.get_registro(x);
746
747         if(auxiliar instanceof Enfermero){
748             Enfermero aux = (Enfermero)auxiliar;
749
750             System.out.println(aux.toString() + "      "+String.format("%-15d",aux.numero_tratamientos())
751             + String.format("%-4d",((Trabajador)auxiliar).num_casos_PS()));
752         }
753     }
754
755     System.out.println("Datos de Administradores: \n");
756     System.out.println(imp());
757     for(int x : recorrido)
758     {
759
760         auxiliar = baseDatos.get_registro(x);
761
762         if(auxiliar instanceof Administrador){
763             Administrador aux = (Administrador)auxiliar;
764             System.out.println(aux.toString() );
765
766         }
767     }
768 }
769
770
771 /**
772 * Metodo que actualiza el stock de vacunas, necesario que el usuario de la clinica en el
773 * momento sea un administrador (medida de seguridad y bloqueo)
774 *
775 * @param administrador de la clinica
776 */
777 public void actualizar_stockVacunas(Administrador t)
778 {
```

```
779     System.out.println("En la nevera hay un total actual de " +baseDatos.numeroVacunas()+" vacunas");
780     System.out.println("Pfizer: " +baseDatos.get_numeroPfizer());
781     System.out.println("Moderna: "+baseDatos.get_numeroModerna());
782     System.out.println("JJ: " +baseDatos.get_numeroJJ());
783     System.out.println("\n Desea ampliar el stock?");
784     if(estatico.pedirConfirmacion())
785     {
786         System.out.println(" Seleccione una opcion:");
787         System.out.println(" * 1 agregar Pfizer");
788         System.out.println(" * 2 agregar Moderna");
789         System.out.println(" * 3 agregar JJ");
790         System.out.println(" * 4 Cancelar");
791
792         switch(estatico.pedirNumero(1,4))
793         {
794             case 1:
795                 System.out.println("Agregando Pfizer, inserte unidades");
796                 baseDatos.agrega_Pfizer(estatico.pedirNumero());
797                 break;
798             case 2:
799                 System.out.println("Agregando Moderna, inserte unidades");
800                 baseDatos.agrega_Moderna(estatico.pedirNumero());
801                 break;
802             case 3:
803                 System.out.println("Agregando JJ, inserte unidades");
804                 baseDatos.agrega_JJ(estatico.pedirNumero());
805                 break;
806             case 4:
807                 System.out.println("Saliendo");
808                 break;
809         }
810     }
811     System.out.println("\n\n x x x x x x \n\n");
812
813 }
814
815
816 private void visualizarVacunas(Administrador t)
817 {
818     System.out.println("por implementar");
819 }
820
821
822
823 /**
824 * Menu de trabajadores, necesita que el usuario de la clinica sea un
825 * trabajador, comprueba el tipo de instancia en tiempo de ejecucion y
826 * aplica los metodos y las funciones propias de la clase final.
827 *
828 * Permite que los enfermeros y tecnicos vean la carga de trabajo que tienen
829 * y que actualicen la informacion acordemente.
830 *
831 * @param operario de la clinica
832 */
833
834 public void menuTrabajador(Trabajador x)
835 {
836     boolean salir = false;
837     do{
838         if(x instanceof Enfermero)
839             System.out.println(" Bienvenido Enfermero ");
840         if(x instanceof Tecnico)
841             System.out.println("seleccionne la operacion que desee ejecutar:");
842
843         System.out.println(" * 1 Visualizacion de datos plan Semanal");
844         System.out.println(" * 2 Visualizacion de tratamientos completados");
845         System.out.println(" * 3 Visualizacion datos Paciente");
846         System.out.println(" * 4 Registro y actualizacion de tratamientos");
847         System.out.println (" * 5 Salir");
848
849         switch(estatico.pedirNumero(1,5))
850         {
851
852             case 1:
853                 x.imprimir_planSemanal();
854                 break;
855             case 2:
856                 x.imprimir_completados();
```

```
857         break;
858     case 3:
859         System.out.println("      *1  Pacientes de tratamientos completados");
860         System.out.println("      *2  Pacientes de plan Semanal");
861         System.out.println("      *3  Paciente de la base de datos");
862     switch(estatico.pedirNumero(1,3))
863     {
864         case 1:
865             x.imprimir_pacientes_Completados();
866             break;
867         case 2:
868             x.imprimir_pacientes_planSemanal();
869             break;
870         case 3:
871             imprimir_datosUsuarioUnitario();
872             break;
873     }
874     break;
875     case 4:
876         if(x instanceof Tecnico)
877         {
878             ((Tecnico)x).TrabajarPruebas();
879         } else if(x instanceof Enfermero)
880         {
881             System.out.println("Desea realizar pruebas a pacientes(1) o vacunar?(2)");
882             switch(estatico.pedirNumero(1,2))
883             {
884                 case 2:
885                     ((Enfermero)x).trabajarVacuna();
886                     break;
887
888                 case 1:
889                     ((Enfermero)x).trabajarPrueba();
890                     break;
891             }
892         }
893     break;
894     case 5:
895         salir = true;
896         break;
897     }
898     }while(!salir);
899 }
900
901 /**
902 * Metodo propio de administrador, muestra informacion general o
903 * de un usuario concreto
904 */
905
906
907     private void visualizarDatos(Administrador t)
908     {
909         if((t instanceof Administrador))
910             System.out.println(" * 1  Ver un glosario general del estado de la clinica ");
911             System.out.println(" * 2  Ver expediente de paciente o trabajador");
912
913
914     switch(estatico.pedirNumero(1,2))
915     {
916         case 1:
917             imprimirDatos();
918             break;
919         case 2:
920             imprimir_datosUsuarioUnitario();
921
922             break;
923
924     }
925
926 }
927
928 /**
929 * metodo que imprime los datos de un usuario concreto, lo implementa el menu de trabajador y el
930 * de administrador, comprueba tipo de objeto en tiempo de ejecucion (tipo de persona) e imprime
931 * la informacion especializada de la clase
932 */
933
934
```

```
935     public void imprimir_datosUsuarioUnitario()
936     {
937         System.out.println("Ingrese el DNI");
938         int auxiliar = estatico.pedirNumero();
939         if(baseDatos.contains_registro(auxiliar))
940         {
941             System.out.println("Encontrado");
942             Person comodin = baseDatos.get_registro(auxiliar);
943             if(comodin instanceof Paciente)
944             {
945                 estatico.print_imp();
946                 System.out.println(comodin);
947                 comodin.imprimir_Tratamientos();
948             }
949
950             if(comodin instanceof Trabajador && !(comodin instanceof Administrador) )
951             {
952                 estatico.print_imp();
953                 System.out.println(comodin);
954                 System.out.println("\n\n XXXXXXXX tratamientos completados XXXXXXXX\n");
955                 comodin.imprimir_Tratamientos();
956                 System.out.println("\n\n XXXXXXXX Plan Semanal XXXXXXXX\n");
957                 ((Trabajador)comodin).imprimir_planSemanal();
958             }
959         } else{
960             System.out.println("No encontrado");
961         }
962     }
963
964
965 /**
966 * Manda la prueba con resultado positivo al modulo de confinamiento.
967 * los parametros sirven para restringir el flujo de informacion y lanzar
968 * fallo en tiempo de ejecucion si un usuario que no es de la clase adecuada
969 * intenta usar el metodo. Este metodo solo esta implementado en la clase Tecnico.
970 */
971
972 public void alerta(Tecnico x, Pruebas y)
973 {
974     confinamiento.alerta(y);
975     System.out.println(" modulo confinamiento: ACKNOWLEDGE");
976 }
977
978
979 /**
980 * Manda una senhal al modulo de vacunacion, si el parametro booleano
981 * es igual a true la senhal es de eliminacion de informacion, si
982 * la senhal es false manda una senhal al modulo para indicarle que
983 * el usuario ha sido ya vacunado y sacarlo de las colas de prioridad
984 * correspondientes. No implementado en la version 0.1
985 *
986 */
987 public void alertaVacuna(Trabajador x, Tratamiento y, boolean z)
988 {
989     if(z)
990         System.out.println("modulo vacunacion, elimine");
991     else System.out.println("modulo vacunacion, acknowlege vacunado");
992 }
993
994 }
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
```

```
1 import java.util.LinkedList;
2 import java.util.Queue;
3 import java.util.Iterator;
4
5 /**
6 * 
7 * Funciones del tecnico:
8 *
9 * -Visualizacion de datos de los pacientes asignados
10 *
11 * -Registro y actualizacion de pruebas diagnosticas y vacunacion
12 * @author (Francisco Sanchez)
13 * @version (0.1)
14 *
15 *
16 */
17
18 public class Tecnico extends Trabajador
19 {
20
21
22 /**
23 * constructor de la clase tecnico
24 * @param DNI el dni identificador del tecnico
25 * @param c clinica, la clinica donde va a trabajar el tecnico, necesario para las alertas
26 */
27 public Tecnico(int DNI,Clinica c)
28 {
29     super(DNI, c);
30     nueva_listaTratamientos();
31     in_planSemanal();
32 }
33
34 /**
35 * constructor completo para las funciones de debug y pruebas. No necesario en el programa final
36 */
37 public Tecnico(Clinica c, int DNI, String nombre, String apellido1, String apellido2, int edad)
38 {
39     super(DNI, c);
40     in_planSemanal();
41     nueva_listaTratamientos();
42     this.set_edad(edad);
43     this.set_nombre(nombre);
44     this.set_apellido1(apellido1);
45     this.set_apellido2(apellido2);
46 }
47
48 /**
49 * metodo sobreescrito para agregar tratamientos a la lista con filtro.
50 * El tecnico no necesita
51 * aceptar vacunas, por que lo que el primer condicional funciona de filtro
52 */
53 @Override
54 public void add_Tratamiento(Tratamiento x)
55 {
56     if(x instanceof Pruebas && x != null)
57         aceptar_Tratamiento(x);
58     else {
59         System.out.println("El tratamiento esta vacio o no es una prueba");
60     }
61 }
62
63 }
64
65 /**
66 * metodo sobreescrito del metodo abstracto de la clase trabajador.
67 * Este metodo comprueba que se pueda aceptar el tratamiento, por tipo
68 * y por capacidad
69 */
70 @Override
71 public void aceptar_Tratamiento(Tratamiento x)
72 {
73     if(x instanceof Pruebas ){
74         if(planSemanal_espacioLibre())
75         {
76             this.agregar_planSemanal((Pruebas)x);
77         }else
78         {
```

```
79         System.out.println("falló en aceptar_Tratamiento // Tecnico");
80     }
81 }else{
82     System.out.println("tipo no aceptado por el tecnico");
83 }
84 }
85
86 /**
87 * Pone al tecnico a trabajar, procesa todas las pruebas de a una.
88 */
89 public void TrabajarPruebas()
90 {
91
92     Iterator x = this.iterator_planSemanal();
93     boolean encontrado = false;
94     Pruebas auxiliar;
95
96     while(x.hasNext())
97     {
98         auxiliar = (Pruebas)x.next();
99         if(auxiliar.get_ejecutado())
100        {
101            auxiliar.analizar_prueba(this);
102            encontrado = true;
103            if( !(auxiliar instanceof AnalisisSerologico) && auxiliar.get_resultado())
104            {
105                this.alerta_Clinica(this, auxiliar);
106            }
107        }
108    }
109
110
111    if(!encontrado) System.out.println("no hay pruebas procesadas que analizar");
112    if(x == null) System.out.println("Se han recorrido las pruebas");
113
114 }
115
116
117 /**
118 * comprueba que el plan semanal tenga espacio para aceptar trabajos con respecto a los limites
correspondientes a
119 * la subclase
120 */
121 @Override
122     public boolean planSemanal_espacioLibre()
123 {
124     return super.planSemanal_espacioLibre(estatico.ENF_LIMITE);
125 }
126
127 /**
128 * comprueba que el plan semanal tenga espacio para aceptar trabajos con respecto a los limites
correspondientes a
129 * un parametro dado
130 * @param x los tratamientos que puede aceptar como limite
131 */
132 @Override
133     public boolean planSemanal_espacioLibre(int x)
134 {
135     return super.planSemanal_espacioLibre(x);
136 }
137
138
139 }
140 }
```

```
1 /**
2  * Clase que representa una nevera donde se almacenan las vacunas
3  *
4  * @author (Francisco Sanchez)
5  * @version (0.1)
6  */
7
8 public class Nevera
9 {
10
11     private int vacunasJJ;
12     private int vacunasPfizer;
13     private int vacunasModerna;
14     private int vacunasTotal;
15
16 /**
17  * Constructor de la clase nevera, inicializa los valores a 0
18  */
19 public Nevera()
20 {
21     vacunasJJ = 0;
22     vacunasPfizer = 0;
23     vacunasModerna = 0;
24     vacunasTotal = 0;
25 }
26
27 /**
28  * funcion que devuelve una vacuna de forma aleatoria de la nevera,
29  * si la que se ha elegido no tiene existencias se vuelve a elejir aleatoriamente,
30  * hasta que al final devuelve una.
31  * @return vacunita vacuna de forma aleatoria
32  */
33 public Vacunas get_vacuna_Random()
34 {
35     Vacunas vacunita = null;
36     if(vacunasTotal >0)
37     {
38         do{
39             switch(estatico.numeroRandom(2))
40             {
41                 case 0:
42                     vacunita = this.get_vacunaJJ();
43                     break;
44                 case 1:
45                     vacunita = this.get_vacunaModerna();
46                     break;
47                 case 2:
48                     vacunita = this.get_vacunaPfizer();
49                     break;
50             }
51
52         }while(vacunita == null);
53         vacunasTotal--;
54     }else
55     {
56         System.out.println("No quedan Vacunas");
57     }
58     return vacunita;
59 }
60
61
62 /**
63  * metodos que devuelven una vacuna de forma no random. No util para el programa
64  * principal, pero si para las pruebas.
65  */
66     public void recibe_JJ(int x)
67     {
68         this.vacunasJJ += x;
69         vacunasTotal += x;
70     }
71     public void recibe_Pfizer(int x)
72     {
73         this.vacunasPfizer += x;
74         vacunasTotal += x;
75     }
76     public void recibe_Moderna(int x)
77     {
78         this.vacunasModerna += x;
```

```
79         vacunasTotal += x;
80     }
81
82     /**
83      * metodo que devuelve el numero de vacunas en total almacenadas en la nevera
84      * @return vacunasTotal
85      */
86     public int get_numeroVacunas()
87     {
88         return vacunasTotal;
89     }
90
91     /**
92      * metodo que devuelve el numero de vacunas en total de un tipo dado
93      * @return vacunasPfizer
94      */
95     public int get_numeroPfizer()
96     {
97         return this.vacunasPfizer;
98     }
99
100    /**
101       * metodo que devuelve el numero de vacunas en total de un tipo dado
102       * @return vacunasModerna
103       */
104     public int get_numeroModerna()
105     {
106         return this.vacunasModerna;
107     }
108
109    /**
110       * metodo que devuelve el numero de vacunas en total de un tipo dado
111       * @return vacunasJJ
112       */
113     public int get_numeroJJ()
114     {
115         return this.vacunasJJ;
116     }
117
118    /**
119       * metodo que devuelve una vacuna especifica, de un tipo dado
120       * @return JJ
121       */
122     public JJ get_vacunaJJ()
123     {
124         if(vacunasJJ >0)
125         {
126             vacunasJJ--;
127             return new JJ();
128         }else
129         {
130             return null;
131         }
132     }
133
134    /**
135       * metodo que devuelve una vacuna especifica, de un tipo dado
136       * @return Pfizer
137       */
138     public Pfizer get_vacunaPfizer()
139     {
140         if(vacunasPfizer >0)
141         {
142             vacunasPfizer--;
143             return new Pfizer();
144         }else
145         {
146             return null;
147         }
148     }
149
150    /**
151       * metodo que devuelve una vacuna especifica, de un tipo dado
152       * @return Moderna
153       */
154     public Moderna get_vacunaModerna()
155     {
156
```

```
157     if(vacunasJJ >0)
158     {
159         vacunasModerna--;
160         return new Moderna();
161     }else
162     {
163         return null;
164     }
165 }
166
167 }
```

```
1 /**
2  * Write a description of class Moderna here.
3  *
4  * @author (Francisco Sanchez)
5  * @version (0.1)
6  */
7
8 public class Moderna extends Vacunas
9 {
10
11    /**
12     * Constructor for objects of class Moderna
13     */
14    public Moderna()
15    {
16        super();
17    }
18
19
20}
21
```

```
1 import java.io.Serializable;
2 import java.time.LocalDate;
3
4 /**
5  * Abstract class Tratamiento - Esta clase representa de forma general cualquier
6  * procedimiento por el cual un paciente o usuario haya acudido a la clinica.
7  * Agrupa todos puesto que cualquiera, ya sea vacuna o prueba necesita almacenar
8  * una informacion muy parecida, utilizando la herencia se ahorra la duplicitad
9  * del codigo en gran medida. Tiene un alto grado de uso de polimorfismo y
10 * de comprobaciones en tiempo de ejecucion para dirigir el curso del programa
11 *
12 *
13 * @author (Francisco Sanchez)
14 * @version (0.1)
15 */
16 public abstract class Tratamiento implements Serializable
17 {
18     private Tecnico tecnicoAsignado;
19     private Enfermero enfermeroAsignado;
20     private Paciente paciente;
21     private LocalDate fecha;
22
23 /**
24  * constructor de la clase Tratamiento, esta es abstracta, aun asi dispone que cualquier
25  * tratamiento que se construya deba tener la fecha del momento "estampada"
26  */
27 public Tratamiento()
28 {
29     fecha = LocalDate.now();
30 }
31
32
33
34 /**
35  * devuelve una cadena con el tipo de subclase assignable al tratamiento
36  * return tipo de clase
37  */
38 public String get_tipoTratamiento()
39 {
40
41     if(this instanceof JJ) return ("vacuna: JJ");
42     if(this instanceof Moderna) return ("vacuna: Moderna");
43     if(this instanceof Pfizer) return ("vacuna: Pfizer");
44     if(this instanceof TestPcr) return ("Test PCR ");
45     if(this instanceof PruebaClasica) return ("Prueba Clasica");
46     if(this instanceof PruebaRapida) return("Prueba Rapida");
47     if(this instanceof AnalisisSerologico) return ("Analisis Serologico");
48     return ("Categoria no identificada");
49 }
50
51 /**
52  * metodo mutador, asigna tecnico a la prueba
53  */
54 public void set_tecnico(Tecnico x)
55 {
56     this.tecnicoAsignado = x;
57 }
58
59 /**
60  * metodo que comprueba si este tratamiento ha sido correctamente
61  * asignado.
62  */
63 public boolean aceptado()
64 {
65     if(!(this instanceof Vacunas)){
66         if(this.enfermeroAsignado != null && this.tecnicoAsignado !=null)
67             return true;
68         else return false;
69
70     }else if(this.enfermeroAsignado != null) return true;
71     else return false;
72 }
73
74 /**
75  * metodo mutador, asigna enfermero a la prueba
76  */
77 public void set_enfermeroAsignado(Enfermero x)
78 {
```

```
79     this.enfermeroAsignado = x;
80 }
81
82 /**
83 * metodo mutador, asigna paciente a la prueba
84 */
85 public void set_paciente(Paciente y)
86 {
87     this.paciente = y;
88 }
89
90 /**
91 * metodo selector, devuelve paciente del tratamiento
92 * @return paciente
93 */
94 public Paciente get_Paciente()
95 {
96     return this.paciente;
97 }
98
99
100 /**
101 * metodo selector, devuelve el identificador del paciente del tratamiento
102 * @return identificadorDNI
103 */
104 public int get_dni_Paciente()
105 {
106     return this.paciente.get_identificadorDNI();
107 }
108
109 /**
110 * metodo selector, devuelve enfermero del tratamiento
111 * @return enfermeroAsignado
112 */
113 public Enfermero get_Enfermero()
114 {
115     return this.enfermeroAsignado;
116 }
117
118 /**
119 * metodo selector, devuelve tecnico del tratamiento
120 * @return tecnicoAsignado
121 */
122 public Tecnico get_Tecnico()
123 {
124     return this.tecnicoAsignado;
125 }
126
127 /**
128 * metodo mutador, remueve tecnico del tratamiento
129 */
130 public void remove_Tecnico()
131 {
132     this.tecnicoAsignado = null;
133 }
134
135 /**
136 * metodo mutador, remueve enfermero del tratamiento
137 */
138 public void remove_Enfermero()
139 {
140     this.enfermeroAsignado = null;
141 }
142
143 /**
144 * metodo mutador, remueve paciente del tratamiento
145 */
146 public void remove_Paciente()
147 {
148     this.paciente = null;
149 }
150
151 /**
152 * metodo que se asegura de que no queda rastro del tratamiento a lo largo del programa
153 * este metodo tiene que revisarse pues contiene bugs, version actual (0.1)
154 */
155 public void remove_Tratamiento()
156 {
```

```
157     while(technicoAsignado != null)
158     {
159         if(technicoAsignado.contains_Tratamiento(this)){
160             technicoAsignado.remove_Tratamiento(this);
161         }else{
162             tecnicoAsignado = null;
163         }
164     }
165     while(enfermeroAsignado != null)
166     {
167         if(enfermeroAsignado.contains_Tratamiento(this)){
168             enfermeroAsignado.remove_Tratamiento(this);
169         }else{
170             enfermeroAsignado = null;
171         }
172     }
173     while(technicoAsignado != null)
174     {
175         if(technicoAsignado.contains_Tratamiento(this))
176         {
177             technicoAsignado.remove_Tratamiento(this);
178         }else{
179             tecnicoAsignado = null;
180         }
181     }
182     while(paciente != null)
183     {
184         if(paciente.contains_Tratamiento(this))
185         {
186             tecnicoAsignado.remove_Tratamiento(this);
187         }else
188         {
189             tecnicoAsignado = null;
190         }
191     }
192 }
193
194 /**
195 * metodo mutador que asigna la fecha actual al tratamiento
196 */
197 public void timeStamp()
198 {
199     fecha = LocalDate.now();
200 }
201
202 /**
203 * metodo selector que devuelve la fecha asignada al tratamiento
204 */
205 public LocalDate get_fecha()
206 {
207     return this.fecha;
208 }
209
210 /**
211 * metodo mutador debug para cambiar la fecha del tratamiento
212 */
213 public void set_fecha(LocalDate debug)
214 {
215     this.fecha = debug;
216 }
217
218 /**
219 * funcion que calcula cuantos dias han pasado desde una fecha hasta la
220 * fecha almacenada en el tratamiento.
221 * Esta funcion es util para calcular los tiempos entre vacunas o tratamientos.
222 */
223 public boolean cuota_tiempo(int days)
224 {
225     LocalDate aux = null;
226     return this.fecha.plusDays(days).isBefore(aux.now());
227 }
228
229
230 /**
231 * metodo interfaz, permite la correcta utilizacion del polimorfismo
232 */
233 abstract public int get_lapsoDias();
234 }
```

235 | }
236 |
237 |

```
1 import java.io.Serializable;
2 import java.util.HashMap;
3 import java.util.LinkedList;
4 import java.util.Set;
5
6
7 /**
8 * La base de datos de la que se nutre la clase clinica y donde se almacena
9 * toda la informacion, implementa la interfaz Serializable para poder hacer
10 * una copia y mandarlo como stream de digitos binarios a un archivo fuera del entorno
11 * de bluej y a la inversa, para cargar un objeto base de datos de un archivo independiente
12 * y restaurar la informacion de una ejecucion anterior.
13 * No implementada esta ultima funcion en la version 0.1.
14 *
15 * Utiliza una estructura HashMap que almacena una tupla de valores <Integer, Person>
16 * para guardar un registro de los usuarios de la clinica, <lleva, valor>.
17 *
18 * Utiliza una instancia de la clase Nevera, que guarda la informacion
19 * del stock de vacunas, como su nombre indica, representa la idea de una
20 * nevera, donde se guardan los distintos tipos de vacunas.
21 *
22 * Tambien hace uso de 4 variables int que guardan un contador de los tipos
23 * de usuarios registrados en la base de datos.
24 *
25 * @author (Francisco Sanchez)
26 * @version (0.1)
27 */
28
29
30
31 public class BBDD implements Serializable
32 {
33     private HashMap<Integer, Person> registro;
34     private LinkedList<Tratamiento> fichero;
35     private Nevera neverita;
36     int numeroTecnicos;
37     int numeroEnfermeros;
38     int numeroPacientes;
39     int numeroAdministradores;
40
41
42
43 /**
44 * Constructor for objects of class BBDD
45 */
46 public BBDD()
47 {
48     this.registro = new HashMap<Integer, Person>();
49     this.fichero = new LinkedList<>();
50     this.neverita = new Nevera();
51
52     numeroTecnicos = 0;
53     numeroEnfermeros = 0;
54     numeroPacientes = 0;
55     numeroAdministradores = 0;
56 }
57
58 public void add_registro(int dni, Person datosPersona)
59 {
60     this.registro.put(dni, datosPersona);
61     if(datosPersona instanceof Tecnico) numeroTecnicos++;
62     if(datosPersona instanceof Enfermero) numeroEnfermeros++;
63     if(datosPersona instanceof Paciente) numeroPacientes++;
64     if(datosPersona instanceof Administrador) numeroAdministradores++;
65 }
66
67 public int numeroTecnicos(){ return numeroTecnicos;}
68 public int numeroEnfermeros(){return numeroEnfermeros;}
69 public int numeroPacientes(){ return numeroPacientes;}
70 public int numeroAdministradores(){ return numeroAdministradores;}
71
72
73 public Person get_registro(int dni)
74 {
75     return this.registro.get(dni);
76 }
77
78 public HashMap get_archivo()
```

```
79     {
80         return this.registro;
81     }
82
83     public boolean contains_registro(int dni)
84     {
85         return registro.containsKey(dni);
86     }
87
88     public void eliminar_registro(int dni)
89     {
90         Person auxiliar;
91         auxiliar = this.registro.get(dni);
92         if(!(auxiliar instanceof Administrador) || numeroAdministradores != 1)
93         {
94             Tratamiento comodin;
95             while(auxiliar.numero_tratamientos() > 0){
96                 comodin = auxiliar.poll_Tratamiento();
97                 comodin.remove_Tratamiento();
98                 //      this.fichero.remove(comodin);
99             }
100
101            if(auxiliar instanceof Tecnico) numeroTecnicos--;
102            if(auxiliar instanceof Enfermero) numeroEnfermeros--;
103            if(auxiliar instanceof Paciente) numeroPacientes--;
104            if(auxiliar instanceof Administrador) numeroAdministradores--;
105
106            this.registro.remove(dni);
107        }else
108        {
109            System.out.println("No se puede eliminar al unico administrador del sistema");
110        }
111    }
112
113    public int size_registro()
114    {
115        return this.registro.size();
116    }
117
118    public void add_Tratamiento(Tratamiento trat)
119    {
120        this.fichero.add(trat);
121    }
122
123    public LinkedList<Vacunas> get_vacunas()
124    {
125        if(!fichero.isEmpty())
126        {
127            LinkedList<Vacunas> vacunas = new LinkedList<>();
128            for(Tratamiento x: fichero)
129            {
130                if(x instanceof Vacunas)
131                {
132                    vacunas.add((Vacunas)x);
133                }
134            }
135            if(!vacunas.isEmpty())
136            {
137                return vacunas;
138            }else
139            {
140                return null;
141            }
142
143        }else
144        {
145            return null;
146        }
147    }
148
149 }
150 /**
151 * metodos accesores del objeto Nevera
152 */
153 public int numeroVacunas()
154 {
155     return this.neverita.get_numeroVacunas();
156 }
```

```
157
158     public JJ get_vacunaJJ()
159     {
160         return this.neverita.get_vacunaJJ();
161     }
162
163     public Moderna get_vacunaModerna()
164     {
165         return this.neverita.get_vacunaModerna();
166     }
167
168     public Pfizer get_vacunaPfizer()
169     {
170         return this.neverita.get_vacunaPfizer();
171     }
172
173     public void agrega_JJ(int x)
174     {
175         this.neverita.recibe_JJ(x);
176     }
177
178     public void agrega_Pfizer(int x)
179     {
180         this.neverita.recibe_Pfizer(x);
181     }
182
183     public void agrega_Moderna(int x)
184     {
185         this.neverita.recibe_Moderna(x);
186     }
187
188     public Nevera get_Nevera()
189     {
190         return this.neverita;
191     }
192
193     public int get_numeroPfizer()
194     {
195         return this.neverita.get_numeroPfizer();
196     }
197
198     public int get_numeroModerna()
199     {
200         return this.neverita.get_numeroModerna();
201     }
202
203     public int get_numeroJJ()
204     {
205         return this.neverita.get_numeroJJ();
206     }
207
208     /**
209      * Devuelve una lista con los pacientes extraidos de la lista original
210      */
211
212     public LinkedList<Paciente> lista_Pacientes()
213     {
214         Person auxiliar;
215         Set<Integer> recorrido = get_archivo().keySet();
216         LinkedList<Paciente> devolver = new LinkedList<>();
217
218         for(int x: recorrido)
219         {
220
221             auxiliar = get_registro(x);
222             if(auxiliar instanceof Paciente){
223                 devolver.add(((Paciente)auxiliar));
224
225             }
226
227         }
228         return devolver;
229     }
230
231     /**
232      * Devuelve una lista con los tecnicos extraidos de la lista original
233      */
234     public LinkedList<Trabajador> lista_Tecnicos()
```

```
235  {
236      Person auxiliar;
237      Set<Integer> recorrido = get_archivo().keySet();
238      LinkedList<Trabajador> devolver = new LinkedList<>();
239
240      for(int x: recorrido)
241      {
242
243          auxiliar = get_registro(x);
244          if(auxiliar instanceof Tecnico){
245              devolver.add(((Tecnico)auxiliar));
246
247          }
248
249      }
250      return devolver;
251  }
252
253
254 /**
255 * Devuelve una lista con los enfermeros extraidos de la lista original
256 */
257 public LinkedList<Trabajador> lista_Enfermero()
258 {
259     Person auxiliar;
260     Set<Integer> recorrido = get_archivo().keySet();
261     LinkedList<Trabajador> devolver = new LinkedList<>();
262
263     for(int x: recorrido)
264     {
265
266         auxiliar = get_registro(x);
267         if(auxiliar instanceof Enfermero){
268             devolver.add(((Enfermero)auxiliar));
269
270         }
271
272     }
273     return devolver;
274 }
275
276
277 /**
278 * saca una vacuna aleatoria de la nevera
279 * @return Vacunas
280 */
281 public Vacunas get_vacuna_Random(){
282     return this.neverita.get_vacuna_Random();
283 }
284
285
286 }
287
288
289 }
```

```
1 import java.util.LinkedList;
2 import java.time.LocalDate;
3 import java.time.temporal.ChronoUnit;
4 import java.util.Collections;
5
6 /**
7 * Esta clase se encarga de los confinamientos, utiliza una
8 * clase privada NodoConfinamiento, compuesta de 3 listas
9 * las cuales van pasandose las pruebas segun se confirme que el
10 * paciente esta aislandose.
11 * Puede extenderse con una funcion que alerte a la policia despues
12 * de tres intentos fallidos de comunicacion o con la inclusion de
13 * comentarios en las pruebas/casos con lo que
14 * se podria utilizar un patron decorador antes de incluir
15 * la prueba en el nodo, sustituyendo estas listas de pruebas por esta
16 * clase decoradora. En esta version no estan
17 * implementadas estas opciones extra.
18 *
19 * Este modulo tiene una lista que va recibiendo alertas y las
20 * almacena en una lista para ser procesadas, la funcion que las procesa
21 * comprueba que el usuario de la prueba alerta no este ya en el sistema
22 * ya que los antigenos se pueden repetir sin restriccion de tiempo.
23 * Si la alerta la a disparado un test serologico (error) lo descarta.
24 * Al procesar satisfactoriamente una alerta la coloca en un nodo "hoy"
25 * listo para ser procesado al final del dia.
26 *
27 * Al finalizar el dia se recorre la lista de nodos, se reinicia la
28 * logica de las listas internas (procesar, no contestado y confirmado
29 * y se eliminan los nodos con una antiguedad mayor de 10 dias con
30 * respeto a la fecha actual.
31 * Estos nodos se almacenan en una lista de nodos para pedir analisis
32 * serologicos post confinamiento.
33 * A continuacion se almacena el nodo "hoy" al principio de la lista.
34 *
35 * @author (Francisco Sanchez)
36 * @version (0.1)
37 */
38 public class ModuloConfinamiento
39 {
40
41     private class NodoConfinamiento implements Comparable<NodoConfinamiento>
42     {
43
44         LinkedList<Pruebas> procesar;
45         LinkedList<Pruebas> no_contestado;
46         LinkedList<Pruebas> completado;
47
48         LocalDate fecha;
49
50         /**
51          * Constructor de clase NodoConfinamiento, imprime una
52          * fecha concreta en el nodo. Esta fecha deberia de ser
53          * inmutable pero por motivos de pruebas y debug se ha
54          * agregado un metodo mutador.
55          */
56         NodoConfinamiento()
57     {
58             procesar = new LinkedList<>();
59             no_contestado = new LinkedList<>();
60             completado = new LinkedList<>();
61             fecha = LocalDate.now();
62
63         }
64
65         /**
66          * agrega una prueba a la lista procesar del nodo
67          * @param la prueba a procesar
68          */
69         private void add_prueba(Pruebas x)
70         {
71             procesar.add(x);
72         }
73
74         /**
75          * metodo para funciones de debug, a eliminar en programa
76          * final
77          */
78         * @param listaDebugPruebas
```

```
79     */
80     private void add_bloquePruebas(LinkedList<Pruebas> x)
81     {
82         for(Pruebas y : x)
83         {
84             procesar.add(y);
85         }
86     }
87
88
89     /**
90      * metodo que imprime la informacion de los pacientes
91      * almacenados en la lista procesar.
92      */
93     private void imprimir_procesar()
94     {
95         System.out.println("XX PACIENTES POR LLAMAR XX");
96         estatico.print_imp();
97         for(Pruebas x : procesar)
98         {
99             System.out.println(x.get_Paciente().toString());
100        }
101    }
102
103 }
104
105 /**
106  * metodo que imprime la informacion de los pacientes
107  * almacenados en la lista de no contestados.
108  */
109 private void imprimir_no_contestado()
110 {
111     if(no_contestado.size() > 0)
112     {
113         System.out.println("XX CALL BACKS XX");
114         estatico.print_imp();
115         for(Pruebas x : no_contestado)
116         {
117             System.out.println(x.get_Paciente().toString());
118         }
119     }
120 }
121
122
123 /**
124  * metodo que imprime la informacion de los pacientes
125  * almacenados en la lista de completados
126  */
127
128 private void imprimir_completados()
129 {
130     if(completado.size() > 0){
131         System.out.println("XX CONTACTO CONFIRMADO XX");
132         estatico.print_imp();
133         for(Pruebas x : completado)
134         {
135             System.out.println(x.get_Paciente().toString());
136         }
137     }
138 }
139
140
141 /**
142  * metodo que imprime un resumen del dia, no tiene en cuenta que las listas
143  * puedan estar vacias, esto se hace en los respectivos metodos
144  */
145 private void imprimir_resumen()
146 {
147     System.out.println("*****");
148     System.out.println("Dia " + dia_Confi() + " de aislamiento ");
149     imprimir_procesar();
150     imprimir_no_contestado();
151     imprimir_completados();
152     System.out.println();
153     System.out.println("*****");
154 }
155
156 /**

```

```
157     * metodo que recorre las listas comprobando la pertenencia
158     * del paciente pasado por parametro en estas, si lo encuentra
159     * devuelve un true.
160     * @return encontrado
161     */
162     private boolean contains(Paciente x)
163     {
164         boolean encontrado = false;
165         int dniPaciente = x.get_identificadorDNI();
166         if(!procesar.isEmpty()){
167             for(Pruebas y: procesar){
168                 if(y.get_Paciente().get_identificadorDNI() == dniPaciente)
169                     encontrado = true;
170             }
171         }
172
173         if(!encontrado && !no_contestado.isEmpty())
174             for(Pruebas y: no_contestado){
175                 if(y.get_Paciente().get_identificadorDNI() == dniPaciente)
176                     encontrado = true;
177             }
178         if(!encontrado && !completado.isEmpty()){
179             for(Pruebas y: completado){
180                 if(y.get_Paciente().get_identificadorDNI() == dniPaciente)
181                     encontrado = true;
182             }
183         }
184         return encontrado;
185     }
186
187     /**
188      * pasa la informacion de las otras listas a la de procesar, dejando
189      * listo el nodo para el dia siguiente.
190     */
191     private void reiniciarDia()
192     {
193         while(!no_contestado.isEmpty())
194         {
195             procesar.add(no_contestado.poll());
196         }
197         while(!completado.isEmpty())
198         {
199             procesar.add(completado.poll());
200         }
201     }
202
203     /**
204      * metodo mutador de fecha, no tiene sentido en el programa principal pero
205      * permite cambiar la fecha a un modulo a la fecha actual en el modo debug
206     */
207     private void set_fecha()
208     {
209         fecha = LocalDate.now();
210     }
211
212     private LocalDate get_fecha()
213     {
214         return this.fecha;
215     }
216
217     /**
218      * metodo mutador de fecha, toma como parametro una fecha concreta.
219      * No tiene sentido en el programa principal pero es util en el modo debug
220      * o pruebas.
221      * @param fecha LocalDate
222     */
223     private void set_fecha_ajustada(LocalDate fecha)
224     {
225         this.fecha = fecha;
226     }
227
228     /**
229      * devuelve la diferencia de dias del nodo con respecto a la fecha actual
230     */
231     private int dia_Confi()
232     {
233         return (int) ChronoUnit.DAYS.between(fecha, LocalDate.now());
234     }
```

```
235
236     /**
237      * metodo logico de la clase nodo, recorre la lista de procesar y
238      * mueve las pruebas con respecto a los datos suministrados
239      */
240     private void llamar_con_confirmacion()
241     {
242         Pruebas aux = null;
243
244         if(!procesar.isEmpty()){
245             aux = procesar.pollLast();
246         }else if(!no_contestado.isEmpty()){
247             aux = no_contestado.pollLast();
248         }
249
250         if(aux != null)
251         {
252             System.out.println("XX LLamar: "+ aux.get_Paciente().nomYape());
253             System.out.println("¿Contacto confirmado? (S)í (N)o");
254             if(estatico.pedirConfirmacion()) completado.add(aux);
255             else no_contestado.add(aux);
256
257         }else{
258             System.out.println("Lista vacia:");
259         }
260     }
261
262
263
264     /**
265      * comprueba si la lista procesar ha sido recorrida
266      */
267     public boolean terminado()
268     {
269         return (procesar.isEmpty());
270     }
271
272     /**
273      * metodo debug
274      */
275     private void procesarBloque()
276     {
277         Pruebas aux = null;
278         while(!procesar.isEmpty()){
279             completado.add(procesar.pollLast());
280         }
281         while(!no_contestado.isEmpty()){
282             completado.add(no_contestado.pollLast());
283         }
284         System.out.println("PROCESADOS");
285     }
286
287     @Override
288     public int compareTo(NodoConfinamiento x)
289     {
290         return this.fecha.compareTo(x.get_fecha());
291     }
292 }
293
294 LinkedList<Pruebas> alertas;
295 LinkedList<NodoConfinamiento> confinamiento;
296 NodoConfinamiento hoy;
297 LinkedList<NodoConfinamiento> pendienteSerologico;
298
299 Clinica clinica;
300
301 /**
302  * Constructor de la clase clinica
303  */
304 ModuloConfinamiento(Clinica clinica)
305 {
306     alertas = new LinkedList<>();
307     confinamiento = new LinkedList<>();
308     hoy = new NodoConfinamiento();
309     pendienteSerologico = new LinkedList<>();
310
311     this.clinica = clinica;
312 }
```

```
313
314 /**
315 * agrega una alerta a la lista de alertas
316 * @param prueba
317 */
318 public void alerta(Pruebas prueba)
319 {
320     alertas.add(prueba);
321 }
322
323 /**
324 * procesa la lista de alertas
325 */
326 public void procesar_alertas()
327 {
328     for(Pruebas x : alertas)
329     {
330         if((x instanceof Antigenos))
331         {
332             if(!(this.contiene_Usuario(x.get_Paciente()))) hoy.add_prueba(x);
333         }else
334         {
335             hoy.add_prueba(x);
336         }
337     }
338     alertas.clear();
339 }
340
341 /**
342 * recorre las listas buscando al usuario pasado por parametro
343 * @return encontrado si ha encontrado al usuario en las listas.
344 */
345 public boolean contiene_Usuario(Paciente y)
346 {
347     boolean encontrado = false;
348     for(NodoConfinamiento x : confinamiento)
349         if(x.contains(y)) encontrado = true;
350
351     if(hoy.contains(y)) encontrado = true;
352
353     if(!encontrado)
354         for(NodoConfinamiento x : pendienteSerologico)
355             if(x.contains(y)) encontrado = true;
356
357     return encontrado;
358 }
359
360
361
362 /**
363 * metodo para pedir analisis serologicos desde el modulo de confinamiento.
364 * procesa la lista al completo
365 * @param administrador de la clinica
366 */
367 public void pedir_Serologico/Administrador administrator)
368 {
369     LinkedList<Pruebas> auxiliar = null;
370     for(NodoConfinamiento x: pendienteSerologico)
371     {
372         for(Pruebas y: x.procesar){
373             clinica.asignarTratamiento(administrador, y.get_Paciente().get_identificadorDNI(),1);
374             System.out.println("analisis pedido desde modulo confinamiento");
375             if(y.aceptado()){
376                 if (auxiliar == null) auxiliar = new LinkedList<>();
377                 auxiliar.add(y);
378             }
379         }
380
381         for(Pruebas y: auxiliar)
382         {
383             pendienteSerologico.remove(y);
384         }
385     }
386 }
387
388 /**
389 * metodo para finalizar el dia, agrega el nodo hoy a la lista y elimina
390 * los nodos con una antiguedad superior a diez dias.
```

```
391  /*
392  public void finalizar_dia()
393  {
394      if(hoy == null)
395      {
396          if(confinamiento.peekFirst().get_fecha().compareTo(LocalDate.now()) == 0)
397              hoy = confinamiento.pollFirst();
398          else if(confinamiento.peekLast().get_fecha().compareTo(LocalDate.now()) == 0)
399              hoy = confinamiento.pollLast();
400          else{ hoy = new NodoConfinamiento();}
401      }
402
403
404      confinamiento.add(hoy);
405      hoy = null;
406
407      if(confinamiento.size() > 0){
408          for(NodoConfinamiento x : confinamiento) x.reiniciarDia();
409      }
410      while(confinamiento.getLast().dia_Confi() > 10){
411          pendienteSerologico.add(confinamiento.removeLast());
412      }
413
414      }
415
416
417 /**
418 * metodo debug, acorta el confinamiento por un valor de dias igual al valor
419 * pasado por parametro. Util para pruebas.
420 */
421 public void finalizar_dia(int diaPrueba)
422 {
423
424     confinamiento.addFirst(hoy);
425     hoy = null;
426     while(confinamiento.getLast().dia_Confi() > diaPrueba){
427         pendienteSerologico.add(confinamiento.removeLast());
428     }
429
430
431     }
432
433 /**
434 * imprime un resumen de los dias de confinamiento
435 */
436 public void imprimir_resumen()
437 {
438     for(NodoConfinamiento m : confinamiento)
439     {
440         m.imprimir_resumen();
441     }
442 }
443
444 /**
445 * imprime un resumen de los nodos en la cola de finalizados de confinamiento;
446 * los que esperan por un analisis serologico.
447 */
448 public void imprimir_serologico()
449 {
450     for(NodoConfinamiento m: pendienteSerologico)
451     {
452         m.reiniciarDia();
453         m.imprimir_procesar();
454     }
455 }
456
457 /**
458 * imprime el numero de alertas almacenadas en la lista a la espera de ser
459 * procesadas
460 */
461 public void imprimir_numeroAlertas()
462 {
463     System.out.println(alertas.size() + " alertas");
464 }
465
466 /**
467 * metodo debug, cambia la fecha del modulo hoy por el valor indicado por parametro
468 * un valor negativo cambia la fecha del nodo en el pasado, un valor positivo lo
```

```
469     * parametriza en el futuro.
470     * @param dias
471     */
472     public void set_fecha_nodo_hoy(int dias)
473     {
474         LocalDate aux = LocalDate.now();
475         if(dias >0) aux.plusDays(dias);
476         if(dias <0) aux.minusDays(dias);
477         hoy.set_fecha_ajustada(aux);
478     }
479 /**
480     * metodo debug, no tiene sentido en la implementacion final del programa pues los
481     * nodos siempre se deberian de incluir en orden cronologico.
482     */
483     public void ordenaListaNodoConfinamiento()
484     {
485         Collections.sort(confinamiento);
486     }
487 /**
488     * Carga las opciones del modulo de Confinamiento, hay un limitante a la hora de
489     * acceder al mutador, como medida extra de seguridad
490     * @param administrador
491     */
492
493     public void moduloConfinamiento(Administrador p)
494     {
495         if(hoy == null)
496         {
497             if(confinamiento.peekFirst().get_fecha().compareTo(LocalDate.now()) == 0)
498                 hoy = confinamiento.pollFirst();
499             else if(confinamiento.peekLast().get_fecha().compareTo(LocalDate.now()) == 0)
500                 hoy = confinamiento.pollLast();
501             else{ hoy = new NodoConfinamiento();}
502         }
503     }
504
505     boolean salir = false;
506
507
508
509     do{
510         System.out.println(" * 1 Numero de alertas      ");
511         System.out.println(" * 2 Procesar alertas      ");
512         System.out.println(" * 3 Imprimir resumen general ");
513         System.out.println(" * 4 Imprimir pendientes de analisis serologico (confinamiento finalizado) ");
514         System.out.println(" * 5 Pedir analisis serologicos  ");
515         System.out.println(" * 6 Finalizar dia (reiniciar para dia siguiente)   ");
516         System.out.println(" * 7 Salir\n");
517
518         switch(estatico.pedirNumero(1,7))
519         {
520             case 1:
521                 this.imprimir_numeroAlertas();
522                 break;
523
524             case 2:
525                 this.procesar_alertas();
526                 break;
527
528             case 3:
529                 this.imprimir_resumen();
530                 break;
531
532             case 4:
533                 this.imprimir_serologico();
534                 break;
535
536             case 5:
537                 if(p instanceof Administrador)
538                     this.pedir_Serologico(p); // administrador
539                 break;
540
541             case 6:
542                 this.finalizar_dia();
543                 break;
544
545             case 7:
546                 salir = true;
```

```
547
548         break;
549     }
550
551     //finalizar_dia(diaPrueba); // int
552
553
554
555 }while(!(salir));
556 }
557
558 /**
559 * metodo debug para llenar los modulos de confinamiento.
560 */
561 public void crear_nodosPrueba(LinkedList<Paciente> listaPacientes, Clinica t)
562 {
563     PruebaRapida auxiliar;
564     for( int x = 1; x <= 8; x++)
565     {
566
567         hoy = new NodoConfinamiento();
568         set_fecha_nodo_hoy(x);
569
570         for(int y = 0; y < 3; y++){
571             auxiliar = new PruebaRapida();
572             auxiliar.set_fecha(hoy.get_fecha());
573             auxiliar.set_resultado(true);
574             auxiliar.set_paciente(listaPacientes.poll());
575             hoy.add_prueba(auxiliar);
576
577         }
578         finalizar_dia();
579     }
580 }
581
582
583
584 }
585
586
587
588 }
```

```
1 /**
2  * Esta clase solo diferencia entre tipos de pruebas de antigenos
3  * y no contiene ninguna diferencia con respecto a la clase padre
4  *
5  * @author (Francisco Sanchez)
6  * @version (0.1)
7  */
8
9 public class PruebaRapida extends Antigenos
10 {
11
12
13 /**
14  * Constructor for objects of class PruebaRapida
15  */
16 public PruebaRapida()
17 {
18
19 }
20
21
22
23 }
24
```

```
1 /**
2  * Esta clase representa a un test PCR, impone un limite en
3  * los dias entre pruebas pero no tiene nada mas especial con respecto
4  * a otra prueba cualquiera de analisis de presencia viral
5  *
6  * @author (Francisco Sanchez)
7  * @version (0.1)
8  */
9
10 public class TestPcr extends Pruebas
11 {
12
13
14     private final int lapsoDias = estatico.PCR;
15
16
17     /**
18      * Constructor for objects of class TestPcr
19      */
20     public TestPcr()
21     {
22
23     }
24
25     /**
26      * implementa el metodo abstracto de la clase padre.
27      * @return lapsoDias
28      */
29     @Override
30     public int get_lapsoDias()
31     {
32         return this.lapsoDias;
33     }
34
35
36
37
38
39 }
40 }
```

```
1 /**
2  * Esta clase representa al administrador de la clinica,
3  * las funciones del administrador estan declaradas en la
4  * clinica y esta comprueba que el usuario es un administrador
5  * para permitir la manipulacion de esta.
6  *
7  * @author (Francisco Sanchez)
8  * @version (0.1)
9  */
10 * Administrador; funciones:
11 *
12 * -Gestion de usuarios: altas, bajas y modificaciones de todas las personas
13 *
14 * -Asignacion de pruebas diagnosticas a enfermeros y tecnicos
15 * y asignacion de vacunaciones a enfermeros
16 *
17 * -Visualizacion de datos de todas las personas registradas en el sistema
18 *
19 * -Visualizacion de pacientes asignados a cada enfermero/tecnico para
20 * pruebas diagnosticas y vacunaciones
21 *
22 * -Gestion de la programacion de pruebas serologicas tras los
23 * confinamientos.
24 *
25 * -Actualizacion del stock de vacunas.
26 *
27 * -Visualizacion de la planificacion tentativa de vacunas, a partir de
28 * los pacientes registrados en un momento determinado
29 *
30 *
31 *
32 *
33 */
34 public class Administrador extends Trabajador
35 {
36
37
38
39
40     public Administrador(int DNI, Clinica c)
41     {
42         super(DNI, c);
43     }
44
45
46     public Administrador(Clinica c, int DNI, String nombre, String apellido1, String apellido2, int edad)
47     {
48         super(DNI, c);
49         this.set_edad(edad);
50         this.set_nombre(nombre);
51         this.set_apellido1(apellido1);
52         this.set_apellido2(apellido2);
53     }
54
55
56     @Override
57     public void aceptar_Tratamiento(Tratamiento x)
58     {
59         System.out.println("Nadie manda al administrador");
60     }
61
62     @Override
63     public boolean planSemanal_espacioLibre()
64     {
65         return false;
66     }
67
68     @Override
69     public boolean planSemanal_espacioLibre(int x)
70     {
71         return false;
72     }
73
74     @Override
75     public void add_Tratamiento(Tratamiento x)
76     {
77         System.out.println("Nadie manda al administrador");
78     }
```

79
80
81
82 }
83

```
1 /**
2  * Write a description of class Pfizer here.
3  *
4  * @author (your name)
5  * @version (a version number or a date)
6  */
7
8 public class Pfizer extends Vacunas
9 {
10
11
12 /**
13  * Constructor for objects of class Pfizer
14  */
15 public Pfizer()
16 {
17     super();
18 }
19
20 }
21
```

```
1 /**
2  * Esta clase solo diferencia entre tipos de pruebas de antigenos
3  * y no contiene ninguna diferencia con respecto a la clase padre
4  *
5  * @author (Francisco Sanchez)
6  * @version (0.1)
7  */
8
9 public class PruebaClasica extends Antigenos
10 {
11
12 /**
13  * Constructor for objects of class PruebaClasica
14  */
15 public PruebaClasica()
16 {
17     super();
18 }
19
20
21
22 }
23
```

```
1 /**
2  * Abstract class Pruebas -
3  * clase de la que derivan las pruebas, esta a su vez
4  * hereda de tratamientos
5  *
6  * consta de 3 atributos mas con respecto a Tratamientos;
7  * resultado, procesado y ejecutado. Respectivamente muestran
8  * si ha dado un resultado positivo o negativo, si ha sido procesado por el
9  * tecnico y ejecutado por el enfermero, por lo tanto el resultado
10 * seria el ultimo atributo en asignarse y solo tendria valor si
11 * los otros dos son true.
12 *
13 * @author (Francisco Sanchez)
14 * @version (0.1)
15 */
16 public abstract class Pruebas extends Tratamiento
17 {
18
19
20     private boolean resultado;
21     private boolean procesado;
22     private boolean ejecutado;
23
24     public Pruebas ()
25     {
26         this.resultado = false;
27         this.procesado = false;
28         this.ejecutado = false;
29     }
30
31 /**
32  * metodo abstracto para implementar por las clases hijas
33  * @return numero de dias entre pruebas
34  */
35 abstract public int get_lapsoDias();
36
37 /**
38  * accesor resultado
39  * @param resultado
40  */
41 public boolean get_resultado()
42 {
43     return this.resultado;
44 }
45
46 /**
47  * mutador resultado y procesado.
48  * @param resultado
49  */
50 public void set_resultado(boolean resultado)
51 {
52     this.procesado = true;
53     this.resultado = resultado;
54 }
55
56 /**
57  * accesor procesado
58  * @return procesado
59  */
60 public boolean get_procesado()
61 {
62     return procesado;
63 }
64
65 /**
66  * accesor ejecutado
67  * @return ejecutado
68  */
69 public boolean get_ejecutado()
70 {
71     return this.ejecutado;
72 }
73
74 /**
75  * mutador de ejecutado. Este es el metodo que debe usarse
76  * para cambiar el estado de una prueba, puesto que es el enfermero el
77  * que tiene que declarar que se ha ejecutado la prueba
78  * @param enfermero que ejecuta la prueba
```

```
79 */  
80 public void hacer_prueba(Enfermero enfermero)  
81 {  
82     this.ejecutado = true;  
83 }  
84  
85 /**  
86 * mutador de resultado, la contraparte del metodo hacer_prueba para el tecnico.  
87 * el resultado deberia de aplicarse con este metodo.  
88 * @param tecnico que realiza la prueba.  
89 */  
90 public void analizar_prueba(Tecnico tecnico)  
91 {  
92     this.set_resultado(estatico.resultadoTest());  
93 }  
94  
95 }  
96 }
```

```
1 import java.io.Serializable;
2 import java.io.BufferedReader;
3 import java.io.InputStreamReader;
4 import java.util.LinkedList;
5
6 /**
7  * Abstract class Person - Superclase, define los atributos de un
8  * objeto abstracto persona
9  *
10 * Atributos principales designados en la clase:
11 * Identificador
12 * Nombre
13 * Apellido
14 * Apellido
15 * Edad
16 * lista de tratamientos (el significado de esta lista varia en funcion de la subclase)
17 *
18 * @author (Francisco Sanchez)
19 * @version (0.1)
20 */
21 public abstract class Person implements Serializable
22 {
23
24     private String nombre;
25     private String apellido1;
26     private String apellido2;
27     private final int identificadorDNI;
28     private int edad;
29     private LinkedList<Tratamiento> tratamientos;
30
31 /**
32  * clase constructora, define solo el dni, que pasa a ser un atributo inmutable
33  */
34 Person(int DNI)
35 {
36     this.nombre = "";
37     this.apellido1 = "";
38     this.apellido2 = "";
39     this.identificadorDNI = DNI;
40     this.edad = 0;
41 }
42
43 /**
44  * accesor
45  * @return identificadorDNI
46  */
47 public int get_identificadorDNI()
48 {
49     return this.identificadorDNI;
50 }
51
52 /**
53  * accesor
54  * @return nombre
55  */
56 public String get_nombre()
57 {
58     return this.nombre;
59 }
60
61 /**
62  * accesor
63  * @return apellido1
64  */
65 public String get_apellido1()
66 {
67     return this.apellido1;
68 }
69
70 /**
71  * accesor
72  * @return apellido2
73  */
74 public String get_apellido2()
75 {
76     return this.apellido2;
77 }
78 }
```

```
79    }
80
81    /**
82     * accesor
83     * @return edad
84     */
85    public int get_edad()
86    {
87        return this.edad;
88    }
89
90    /**
91     * mutador, asigna una nueva lista de tratamientos
92     */
93    public void nueva_listaTratamientos()
94    {
95        this.tratamientos = new LinkedList<>();
96    }
97
98    /**
99     * mutador, asigna nombre
100    * @param nombre
101    */
102   public void set_nombre(String nombre)
103   {
104       this.nombre = nombre.trim();
105   }
106
107   /**
108    * mutador, asigna apellido
109    * @param apellido
110    */
111   public void set_apellido1(String apellido)
112   {
113       this.apellido1 = apellido.trim();
114   }
115
116   /**
117    * mutador, asigna segundo apellido
118    * @param apellido
119    */
120   public void set_apellido2(String apellido)
121   {
122       this.apellido2 = apellido.trim();
123   }
124
125   /**
126    * mutador, asigna edad
127    * @param edad
128    */
129   public void set_edad(int edad)
130   {
131       this.edad = edad;
132   }
133
134   /**
135    * accesor, sobrescribe la funcion toString
136    * devuelve una cadena de los atributos de la persona
137    */
138   @Override
139   public String toString()
140   {
141       String comodin = String.format("%-15s",this.nombre) +
142           String.format("%-15s", this.apellido1) +
143           String.format("%-15s", this.apellido2) +
144           String.format("%-5d", this.edad) +
145           String.format("%-8d", this.identificadorDNI);
146       return comodin;
147   }
148
149   /**
150    * funcion mutadora, pide al usuario que introduzca los
151    * datos de la nueva persona creada
152    */
153   public void formularioNuevoRegistro()
154   {
155       do
```

```
157 |     {
158 |
159 |
160 |         System.out.println("introduzca nombre:");
161 |         this.nombre = estatico.pedirTexto();
162 |
163 |         System.out.println("introduzca primer apellido:");
164 |         this.apellido1 = estatico.pedirTexto();
165 |
166 |         System.out.println("introduzca segundo apellido:");
167 |         this.apellido2 = estatico.pedirTexto();
168 |
169 |         System.out.println("introduzca edad:");
170 |         this.edad = estatico.pedirNumero(1,110);
171 |
172 |
173 |         System.out.println("Informacion correcta? (S)i");
174 |
175 |
176 |     }
177 |     while(!estatico.pedirConfirmacion());
178 |
179 |
180 |
181 | /**
182 | * devuelve la lista de tratamientos
183 | * @return tratamientos
184 | */
185 | public LinkedList<Tratamiento> get_tratamientos()
186 | {
187 |     return tratamientos;
188 | }
189 |
190 | /**
191 | * elimina y devuelve un elemento del principio de la lista de tratamientos
192 | * @return Tratamiento
193 | */
194 | public Tratamiento poll_Tratamiento()
195 | {
196 |     return tratamientos.poll();
197 | }
198 |
199 | /**
200 | * mutador, agrega un tratamiento a la lista de tratamientos
201 | * @param x tratamiento
202 | */
203 | public void add_Tratamiento(Tratamiento x)
204 | {
205 |     tratamientos.add(x);
206 | }
207 |
208 | /**
209 | * accesor, devuelve el valor de tamaño de la lista tratamientos
210 | * @return size
211 | */
212 | public int numero_tratamientos()
213 | {
214 |     return this.tratamientos.size();
215 | }
216 |
217 | /**
218 | * mutador, elimina un tratamiento de la lista de tratamientos
219 | * @param x tratamiento
220 | */
221 | public void remove_Tratamiento(Tratamiento x)
222 | {
223 |     tratamientos.remove(x);
224 | }
225 |
226 |
227 | /**
228 | * accesor, comprueba si el tratamiento dado como parametro esta contenido en la
229 | * lista de tratamientos de la persona
230 | * @param x tratamiento
231 | * @return boolean, contiene
232 | */
233 | public boolean contains_Tratamiento(Tratamiento x)
234 | {
```

```
235         return tratamientos.contains(x);
236     }
237
238    /**
239     * funcion que imprime los datos almacenados en la lista de tratamientos
240     */
241    public void imprimir_Tratamientos()
242    {
243        String e = "      ";
244        for(Tratamiento x : this.tratamientos)
245        {
246            System.out.println("XXXX" + x.get_tipoTratamiento() + "XXXX");
247            System.out.println("Fecha      Paciente      Enfermero      Tecnico      ");
248            System.out.print(x.get_fecha() + e+e + x.get_Paciente().get_identificadorDNI() +e+
x.get_Enfermero().get_identificadorDNI() + e+ x.get_Tecnico().get_identificadorDNI());
249            if(!(x instanceof Vacunas)){
250                if(((Pruebas)x).get_ejecutado()) System.out.print("Ejecutado      "); else System.out.print("No
ejecutado      \n");
251                if(((Pruebas)x).get_procesado()) System.out.print("Procesado      "); else System.out.print("No
procesado      \n");
252                if(((Pruebas)x).get_procesado() && ((Pruebas)x).get_ejecutado())
System.out.print(((Pruebas)x).get_resultado() + "\n\n");
253                System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n");
254            }else{
255                if(!(x instanceof JJ)){
256                    System.out.print(((Vacunas)x).get_dosisRestantes() +"  dosis restantes");
257                }
258            }
259        }
260    }
261 }
262 }
263
264 }
```

```
1 import java.util.LinkedList;
2
3 /**
4 *
5 *
6 * @author (Francisco Sanchez)
7 * @version (0.1)
8 *
9 *
10 * Funciones del enfermero:
11 *
12 * -Visualizacion de datos de los pacientes asignados
13 *
14 * -Registro y actualizacion de pruebas diagnosticas y vacunacion
15 *
16 */
17
18 public class Enfermero extends Trabajador
19 {
20
21     private LinkedList<Vacunas> vacunas;
22
23
24 /**
25 * Constructor de la clase enfermero, requiere el identificador unico
26 * en la clinica como lugar de trabajo
27 * @param DNI identificador
28 * @param clinica lugar de trabajo
29 */
30     public Enfermero(int DNI, Clinica clinica)
31     {
32         super(DNI, clinica);
33         in_planSemanal();
34         nueva_listaTratamientos();
35         vacunas = new LinkedList<>();
36     }
37
38 /**
39 * Constructor sobrecargado para crear un enfermero con todos los datos
40 * suministrado por parametro, sin uso en el programa final pero util para
41 * las opciones de debug y pruebas.
42 *
43 * @param clinica lugar de trabajo
44 * @param DNI identificador unico
45 * @param nombre nombre del enfermero
46 * @param apellido1 apellido del enfermero
47 * @param apellido2 segundo apellido del enfermero
48 * @param edad edad del enfermero
49 *
50 */
51
52     public Enfermero(Clinica clinica, int DNI, String nombre, String apellido1, String apellido2, int edad)
53     {
54         super(DNI, clinica);
55         in_planSemanal();
56         nueva_listaTratamientos();
57         vacunas = new LinkedList<>();
58         this.set_edad(edad);
59         this.set_nombre(nombre);
60         this.set_apellido1(apellido1);
61         this.set_apellido2(apellido2);
62     }
63
64 /**
65 * Devuelve el tamano de la lista de vacunaciones
66 * @return size
67 */
68     public int vacunacionesRestantes()
69     {
70         return vacunas.size();
71     }
72
73 /**
74 * Implementacion de la accion de vacunar a un paciente, necesita como parametro una
75 * vacuna que no este vacia, en base a los valores de esta procede a eliminarla
76 * de la lista de vacunas pendientes. En la actualidad no comprueba la coherencia
77
78 */
```

```
79     * con el estado de vacunacion del paciente y esto podria ser una opcion valida a
80     * implementar en el futuro. actual version (0.1)
81     */
82
83     public void vacunar(Vacunas z)
84     {
85         Person x = z.get_Paciente();
86         if(x instanceof Paciente)
87         {
88
89             ((Paciente)x).vacunar(z);
90             if(z.get_dosisRestantes() == 0)
91             {
92                 this.vacunas.remove(z);
93                 this.add_Tratamiento(z);
94                 System.out.println("Inmunizado! Vacunado por completo!");
95             } else{ System.out.println("Primera dosis!"); }
96
97         }else
98         {
99             System.out.println("Todavia no se puede vacunar a no pacientes");
100        }
101    }
102
103 }
104
105 /**
106 * Trabaja la lista de trabajos de vacunacion por completo, comprueba la
107 * posibilidad de vacunar antes de proceder
108 * @pre vacuna no este vacio, paciente pueda vacunarse
109 */
110
111     public void trabajarVacuna()
112     {
113         if(!this.vacunas.isEmpty())
114         {
115             for(Vacunas x : vacunas)
116             {
117                 if(x.get_Paciente().puede_Vacuna()) this.vacunar(x);
118             }
119         }else{ System.out.println("No hay vacunas");}
120     }
121
122 /**
123 * Metodo que formaliza la accion de realizar la prueba, cambia el estado
124 * de la prueba a ejecutado, podria haber sido sobrecargado para aceptar
125 * polimorfismo con respecto al parametro (al trabajador)
126 * pero por semanticas me ha parecido que era mejor mantener
127 * los metodos separados, el de realizar la prueba en si y el de analizarla.
128 * El metodo esta sobreescrito en cada subclase y se accede al apropiado
129 * en tiempo de ejecucion.
130 *
131 * version (0.1)
132 * @param pruebaApaciente una prueba del enfermero
133 * @param enfermero el segundo parametro se contempla en el metodo de la prueba.
134 *          requiere de enfermero como parametro para poder realizar la prueba.
135 */
136     public void realizar_Prueba(Pruebas pruebaApaciente)
137     {
138         pruebaApaciente.hacer_prueba(this);
139
140     }
141
142 }
143
144 /**
145 * metodo sobrecargado de la clase padre persona. Como el enfermero tiene una lista
146 * de vacunas es necesaria la posibilidad de borrar los elementos de esta lista tambien
147 * y de mandar una alerta al modulo de vacunacion para buscar la ficha en este modulo
148 * y poder eliminarla acordemente. En esta version el modulo de vacunacion no esta
149 * completamente terminado.
150 * version (0.1)
151 * @param
152 */
153     @Override
154     public void remove_Tratamiento(Tratamiento x)
155     {
156         super.remove_Tratamiento(x);
157         if(x instanceof Vacunas){
```

```
157     vacunas.remove(x);
158     this.alerta_Clinica(this,x);
159 }
160 }
161 /**
162 * metodo que hace trabajar al enfermero en su lista de trabajos semanal
163 */
164 public void trabajarPrueba()
165 {
166     if(this.tiene_Trabajo())
167     {
168         Pruebas auxiliar = this.coger_prueba();
169         if(auxiliar !=null){
170             this.realizar_Prueba(auxiliar);
171
172         }else {System.out.println("No se ha devuelto un trabajo aunque la funion tiene trabajo"
173                 + "ha devuelto un positivo");
174         }
175     } else
176     {
177         System.out.println("No hay pruebas que ejecutar");
178     }
179 }
180 }
181 }
182 /**
183 * Metodo sobrecargado de la clase padre Trabajador.
184 * El enfermero tiene dos colas de tratamientos, una para las vacunas y otra
185 * para el plan semanal. Este metodo deberia de lanzar una excepcion que pueda
186 * ser manejada por la funcion llamante en caso de que el enfermero rechaze la orden
187 * En la version actual esta funcion no esta implementada. Deberia de hacerse las pertinentes
188 * comprobaciones antes de hacer uso de esta funcion.
189 * @throw noImplementado
190 * @param x tratamiento a aceptar.
191 */
192 @Override
193 public void add_Tratamiento(Tratamiento x)
194 {
195     if(x != null)
196     {
197         aceptar_Tratamiento(x);
198     }else{
199         System.out.println("No hay un tratamiento creado");
200     }
201 }
202 /**
203 * metodo sobreescrito de la funcion padre, aplica las restricciones
204 * pertinentes a la clase enfermero, valor almacenado en clase estatico
205 */
206
207 @Override
208 public boolean planSemanal_espacioLibre()
209 {
210     return super.planSemanal_espacioLibre(estatico.ENF_LIMITE);
211 }
212
213 /**
214 * metodo sobreescrito, aplica una restriccion diferente a los limites de trabajo
215 */
216 @Override
217 public boolean planSemanal_espacioLibre(int x)
218 {
219     return super.planSemanal_espacioLibre(x);
220 }
221
222 /**
223 * metodo sobreescrito de la clase padre, hace comprobaciones extra y redirige
224 * el tratamiento a la lista adecuada
225 */
226 @Override
227 public void aceptar_Tratamiento(Tratamiento x)
228 {
229     if(x instanceof Pruebas){
230         if(planSemanal_espacioLibre() )
231         {
232             this.agregar_planSemanal((Pruebas)x);
```

```
235     }else{
236         System.out.println("No hay espacio libre para pruebas en el plan de este enfermero");
237     }
238     }else{
239         vacunas.add((Vacunas)x);
240     }
241 }
242 }
```

```
1 import java.io.BufferedReader;
2 import java.io.InputStreamReader;
3
4 /**
5  * Clase que almacena funciones, metodos y parametros de utilidad general
6  * en el programa. Permite centralizar y parametrizar el programa de acorde
7  * a las exigencias que se puedan presentar, como un cambio en el tiempo entre
8  * vacunas
9  *
10 * @author (Francisco Sanchez)
11 * @version (0.1)
12 */
13 public abstract class estatico
14 {
15     // TIEMPOS ENTRE PRUEBAS Y VACUNAS
16     static final int TIEMPOVACUNAS = 21;
17     static final int PCR = 15;
18     static final int SEROLOGICO = 180;
19     static final int NUSUARIOTEST = 100;
20     static final int TEC_LIMITE = 4;
21     static final int ENF_LIMITE = 5;
22     static final int PROBABILIDADNEGATIVO = 7; // *10 (porcentaje de negativo)
23
24 /**
25  * constructor, privado, esta clase no representa ningun objeto en si misma
26  */
27 private estatico()
28 {
29 }
30
31 /**
32  * funcion que devuelve un resultado positivo o negativo
33  * en base a un numero random y a 3 posibilidades entre 10,
34  * representando la probabilidad que tienes de dar positivo en el test
35  */
36 public static boolean resultadoTest()
37 {
38     if(Math.random()*10 <= PROBABILIDADNEGATIVO){
39         return false;
40     }
41     return true;
42 }
43
44 /**
45  * Devuelve el resultado de anticuerpos en base a 100
46  * @return int anticuerpos
47  */
48 public static int resultadoAnticuerpos()
49 {
50     return (int)(Math.random()* 100);
51 }
52
53 /**
54  * generador de numeros aleatorios
55  * @param ventana indica la ultima cifra generable; 0 es posibilidad
56  * @return numeroRandom
57  */
58 public static int numeroRandom(int ventana)
59 {
60     return (int)(Math.random()*ventana);
61 }
62
63 static String[] opcionesNombre = {
64     "Pepe", "Maria", "Asuncion", "Gloria", "Lucia", "Lucifer", "Amandonasio", "Teocrido", "Juan", "Jhon10"
65 };
66
67 static String[] opcionesApellidos =
68 {
69     "Sanchez", "Jimenez", "De la Vega", "Ramirez", "Ticchi", "Vecchio", "XXXXdelta", "Omicron", "Hyperion",
70     "Asimov", "Perez", "Reverte",
71 };
72 /**
73  * metodo de apoyo para el modo de test (o debug)
74  * Carga usuarios en una base de datos, se apoya de las funciones de generacion
75  * de numeros aleatorios de la clase estatico
76  * @return baseDatos
77 */
```

```
78  public static BBDD carga_bbdd_prueba(Clinica ClinicaSanPedro)
79  {
80      BBDD baseDatos = new BBDD();
81      baseDatos.add_registro(75910907, ((Person) new Administrador(ClinicaSanPedro,
82                                     75910907, "Francisco", "Sanchez", "Jimenez", 28)));
83      int auxiliar;
84      int auxiliar2;
85
86
87
88
89
90
91      for(int x = 1; x <= estatico.NUSUARIOTEST; x++)
92      {
93          auxiliar2 = (estatico.numeroRandom(3));
94          auxiliar = estatico.numeroRandom(99999999);
95          switch(auxiliar2){
96
97              case 1:
98                  baseDatos.add_registro(auxiliar, ((Person) new Paciente(auxiliar,
99                                     opcionesNombre[estatico.numeroRandom(8)],
100                                    opcionesApellidos[estatico.numeroRandom(22)], opcionesApellidos[estatico.numeroRandom(22)],
101                                    estatico.numeroRandom(110))));
102                  System.out.println("Creado Paciente");
103                  break;
104              default:
105                  baseDatos.add_registro(auxiliar, ((Person) new Tecnico(ClinicaSanPedro, auxiliar,
106                                     opcionesNombre[estatico.numeroRandom(8)],
107                                     opcionesApellidos[estatico.numeroRandom(22)], opcionesApellidos[estatico.numeroRandom(22)],
108                                     estatico.numeroRandom(60))));
109                  System.out.println("Creado Tecnico");
110                  break;
111              case 0 :
112                  baseDatos.add_registro(auxiliar,
113                                     ((Person) new Enfermero(ClinicaSanPedro, auxiliar,
114                                     opcionesNombre[estatico.numeroRandom(8)], opcionesApellidos[estatico.numeroRandom(22)],
115                                     opcionesApellidos[estatico.numeroRandom(22)], estatico.numeroRandom(60)));
116                  System.out.println("Creado Enfermero");
117                  break;
118          }
119
120
121      }
122      return baseDatos;
123  }
124
125
126  public static void cargaPacientes(Clinica ClinicaSanPedro, BBDD baseDatos, int multiplicador)
127  {
128      int limite = 5 * multiplicador;
129      for(int x = 0 ; x <= limite; x++)
130      {
131          System.out.println(x + "dfjakdjfkajdfjaksdjf");
132          baseDatos.add_registro(x, ((Person) new Paciente(x,
133                                         opcionesNombre[estatico.numeroRandom(8)],
134                                         opcionesApellidos[estatico.numeroRandom(22)], opcionesApellidos[estatico.numeroRandom(22)],
135                                         estatico.numeroRandom(110))));
136          System.out.println("Creado Paciente");
137      }
138
139
140
141
142  public static void cargaEnfermeros(Clinica ClinicaSanPedro, BBDD baseDatos, int multiplicador)
143  {
144      int limite = 11 * multiplicador;
145      for(int x = 6 * multiplicador; x <= limite; x++)
146      {
147          baseDatos.add_registro(x,
148                             ((Person) new Enfermero(ClinicaSanPedro, x,
149                                         opcionesNombre[estatico.numeroRandom(8)], opcionesApellidos[estatico.numeroRandom(22)],
150                                         opcionesApellidos[estatico.numeroRandom(22)], estatico.numeroRandom(60)));
151          System.out.println("Creado Enfermero");
152      }
153
154
155 }
```

```
156
157     public static void cargaTecnicos(Clinica ClinicaSanPedro, BBDD baseDatos, int multiplicador)
158     {
159         int limite = 16 * multiplicador;
160         for(int x = 12 * multiplicador; x <= limite; x++)
161         {
162             baseDatos.add_registro(x,
163             ((Person) new Tecnico(ClinicaSanPedro, x,
164             opcionesNombre[estatico.numeroRandom(8)], opcionesApellidos[estatico.numeroRandom(22)],
165             opcionesApellidos[estatico.numeroRandom(22)], estatico.numeroRandom(60))));
166             System.out.println("Creado Tecnico");
167         }
168
169     }
170
171
172     public static void cargaPruebasBBDD(Clinica ClinicaSanPedro, BBDD baseDatos)
173     {
174
175
176         for(int x = 0 ; x <= 9; x++)
177         {
178
179             baseDatos.add_registro(x, ((Person) new Paciente(x,
180             opcionesNombre[estatico.numeroRandom(8)],
181             opcionesApellidos[estatico.numeroRandom(22)], opcionesApellidos[estatico.numeroRandom(22)],
182             estatico.numeroRandom(110)));
183             System.out.println("Creado Paciente");
184
185         }
186
187
188         for(int y = 10; y <= 19; y++)
189         {
190
191             baseDatos.add_registro(y,
192             ((Person) new Enfermero(ClinicaSanPedro, y,
193             opcionesNombre[estatico.numeroRandom(8)], opcionesApellidos[estatico.numeroRandom(22)],
194             opcionesApellidos[estatico.numeroRandom(22)], estatico.numeroRandom(60)));
195             System.out.println("Creado Enfermero");
196
197
198         for(int z = 20; z <= 29; z++)
199         {
200
201             baseDatos.add_registro(z,
202             ((Person) new Tecnico(ClinicaSanPedro, z,
203             opcionesNombre[estatico.numeroRandom(8)], opcionesApellidos[estatico.numeroRandom(22)],
204             opcionesApellidos[estatico.numeroRandom(22)], estatico.numeroRandom(60)));
205             System.out.println("Creado Tecnico");
206
207
208         for(int k = 0; k <= 9; k++)
209         {
210             Paciente uno;
211             Enfermero dos;
212             Tecnico tres;
213             Administrador cuatro = (Administrador)baseDatos.get_registro(75910907);
214
215             uno = (Paciente)baseDatos.get_registro(k);
216             dos = (Enfermero)baseDatos.get_registro(k+10);
217             tres = (Tecnico)baseDatos.get_registro(k+20);
218
219
220
221             //asignar_prueba(int tipo, Paciente x, Administrador y, Tecnico z, Enfermero l)
222             ClinicaSanPedro.asignar_prueba( 2 , uno , cuatro, tres, dos);
223
224
225
226             for(int t = 10; t <= 19; t++)
227             {
228                 ((Enfermero)baseDatos.get_registro(t)).trabajarPrueba();
229             }
230
231             for(int w = 20; w <= 29; w++)
232             {
233
```

```
234         ((Tecnico)baseDatos.get_registro(w)).TrabajarPruebas();  
235     }  
236     }  
237     }  
238     }  
239  
240  
241  
242  
243     /**  
244      * Funcion que pide una secuencia numerica a un usuario y la suministra al programa, esta version no tiene  
245      restriccion en cuanto  
246      * rango de numeros, mientras sea un numero.  
247      * @return entradaInt el numero introducido por el usuario  
248      */  
249     public static int pedirNumero(){  
250         String entrada;  
251         int entradaInt = -1;  
252         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
253         boolean problema = false;  
254         do  
255         {  
256             try{  
257                 entradaInt = Integer.parseInt(br.readLine());  
258                 problema = false;  
259             }catch(Exception e)  
260             {  
261                 System.out.println("no se ha registrado una secuencia numerica");  
262                 System.out.println("por favor, intentelo de nuevo");  
263                 problema = true;  
264             }  
265         }while(problema || entradaInt <= 0 );  
266         return entradaInt;  
267     }  
268  
269     /**  
270      * Funcion que pide una secuencia numerica a un usuario y la suministra al programa, esta version tiene  
271      restriccion en cuanto  
272      * rango de numeros, primer valor representa el inicio y el segundo el final.  
273      * @return entradaInt el numero introducido por el usuario  
274      */  
275     public static int pedirNumero(int inicio, int ultimo){  
276         String entrada;  
277         int entradaInt = -1;  
278         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
279         boolean problema = false;  
280         do{  
281             try{  
282                 entradaInt = Integer.parseInt(br.readLine());  
283                 problema = false;  
284             }catch(Exception e){  
285                 System.out.println("no se ha registrado una secuencia numerica");  
286                 System.out.println("por favor, intentelo de nuevo");  
287                 problema = true;  
288             }  
289         }while(problema || entradaInt < inicio || entradaInt > ultimo );  
290         return entradaInt;  
291     }  
292  
293     /**  
294      * imprime un marco para la visualizacion posterior de datos persona  
295      */  
296     public static void print_imp()  
297     {  
298         System.out.println(String.format("%-15s","Nombre")+String.format("%-15s","Apellido")  
+String.format("%-15s","Apellido")+String.format("%-5s", "edad")  
+String.format("%-8s","DNI"));  
299     }  
300  
301  
302     /**  
303      * Pide una cadena de caracteres al usuario para suministrarla en otros puntos de programa,  
304      * como en nombres o apellidos. Esta version no tiene restricciones en la actual version (0.1)  
305      */  
306     public static String pedirTexto()  
307     {  
308 }
```

```
309     boolean problema = false;
310     String cadena = "no se ha registrado entrada//fallo";
311     BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
312     do{
313
314         try{
315             cadena = br.readLine();
316         }catch(Exception e){
317             System.out.println("Ha habido algun fallo de entrada");
318             System.out.println("por favor, intentelo de nuevo");
319             problema = true;
320         }
321     }while(problema);
322     return cadena;
323 }
324
325 /**
326 * Pide una confirmacion al usuario, devuelve true o false.
327 * @return boolean confirmacion
328 */
329
330 public static boolean pedirConfirmacion()
331 {
332     char auxiliar = 'e';
333     String aux = "";
334     boolean procesado = false;
335     do
336     {
337
338         if(procesado)System.out.print("s? / n?");
339         aux = estatico.pedirTexto().toLowerCase();
340         aux = aux + 'f';
341         auxiliar = aux.charAt(0);
342         procesado = true;
343
344     }
345     while(!(auxiliar != 's' || auxiliar != 'n'));
346     if(auxiliar == 's') {return true;}
347     else {return false;}
348 }
349
350 }
```

```
1 import java.util.LinkedList;
2 import java.util.Iterator;
3
4 /**
5  * Abstract class Trabajador - write a description of the class here
6  *
7  * @author (Francisco Sanchez)
8  * @version (0.1)
9 */
10 public abstract class Trabajador extends Person
11 {
12     private LinkedList<Pruebas> planSemanal;
13     private Clinica clini;
14
15     Trabajador(int DNI, Clinica clinica)
16     {
17         super(DNI);
18         clini = clinica;
19     }
20 }
21
22 /**
23  * metodos interfaz para poder usar polimorfismo, sobreescritos en clases hijas
24 */
25 public abstract void aceptar_Tratamiento(Tratamiento t) ;
26 public abstract boolean planSemanal_espacioLibre();
27
28 /**
29  * mutador constructor de atributo planSemanal, representativo de la carga de trabajo
30  * correspondiente al trabajador
31 */
32 public void in_planSemanal()
33 {
34     this.planSemanal = new LinkedList<>();
35 }
36
37 /**
38  * imprime el plan semanal
39 */
40 public void imprimir_planSemanal()
41 {
42     this.imprimir_Lista(this.planSemanal);
43 }
44
45 /**
46  * imprime los tratamientos almacenados en la lista Tratamientos
47  * declarada en la clase persona. En los pacientes representa el historial
48  * de procedimientos a los que se ha sometido, en los trabajadores los trabajos efectuados.
49  * Otro tipo de historial.
50 */
51 public void imprimir_completados()
52 {
53     this.imprimir_Lista(this.get_tratamientos());
54 }
55
56 /**
57  * imprime la informacion de los pacientes de los tratamientos completados.
58 */
59 public void imprimir_pacientes_Completados()
60 {
61     this.imprimir_pacientes_Tratamientos(this.get_tratamientos());
62 }
63
64 /**
65  * imprime la informacion de los pacientes de los tratamientos del plan semanal
66 */
67 public void imprimir_pacientes_planSemanal()
68 {
69     this.imprimir_pacientes_Tratamientos(this.planSemanal);
70 }
71
72 /**
73  * devuelve el tamnho de la lista del plan semanal, util para compararlo
74  * con los limites impuestos en la practica
75 */
76 public int num_casos_PS()
77 {
78     return this.planSemanal.size();
```

```
79     }
80
81     /**
82      * clase general para imprimir una lista, consta de comprobadores de
83      * tipo en tiempo de ejecucion
84      */
85     private void imprimir_Lista(LinkedList<? extends Tratamiento> p)
86     {
87         String e = "      ";
88         for(Tratamiento x : p)
89         {
90             System.out.println("XXXX" + x.get_tipoTratamiento() + "XXXX");
91             System.out.println("Fecha      Paciente      Enfermero      Tecnico      ");
92             System.out.println(x.get_fecha() + e + x.get_Paciente() + e+ x.get_Enfermero() + e+
x.get_Tecnico());
93             if(!(x instanceof Vacunas)){
94                 if(((Pruebas)x).get_ejecutado()) System.out.print("Ejecutado      "); else System.out.print("No
ejecutado      ");
95                 if(((Pruebas)x).get_procesado()) System.out.print("Procesado      "); else System.out.print("No
procesado      ");
96                 if(((Pruebas)x).get_procesado() && ((Pruebas)x).get_ejecutado())
System.out.print(((Pruebas)x).get_resultado());
97             }else{
98                 if(!(x instanceof JJ)){
99                     System.out.print(((Vacunas)x).get_dosisRestantes() +" dosis restantes");
100                }
101            }
102        }
103    }
104
105    /**
106     * clase general para imprimir la informacion de los pacientes de una lista de tratamientos
107     */
108     private void imprimir_pacientes_Tratamientos(LinkedList<? extends Tratamiento> p)
109     {
110         for(Tratamiento xa : p){
111             System.out.println("XXXX" + xa.get_tipoTratamiento() + "XXXX");
112             estatico.print_imp();
113             System.out.println(xa.get_Paciente().toString());
114         }
115     }
116 }
117
118 /**
119  * iterador para la lista de plan semanal, permite recorrer y visualizar la
120  * lista sin alterar su composicion u orden.
121  */
122 public Iterator iterador_planSemanal()
123 {
124     return planSemanal.listIterator();
125 }
126
127 /**
128  * devuelve la comprobacion de si el objeto trabajador tiene
129  * espacio en su lista de trabajos semanal. Comprueba si alguna creada,
130  * si no la crea. Comprueba si la carga semanal es menor que el limite y
131  * devuelve que hay espacio libre en caso afirmativo. En el caso de que
132  * siga sin haber devuelto un valor true recorre la lista y comprueba los
133  * dias que han pasado entre pruebas, si hay alguna mayor de 7 dias de antiguedad
134  * taponando la lista, la elimina y la agrega a la lista de tratamientos completados.
135  * @param limite de tratamientos por semana
136  * @return puede aceptar otro tratamiento.
137  */
138 public boolean planSemanal_espacioLibre(int limite)
139 {
140     if(this instanceof Administrador) return false;
141     Tratamiento auxiliar = null;
142     if(this.planSemanal == null){
143         planSemanal = new LinkedList<>();
144         return true;
145     }else
146     {
147         if(this.planSemanal.size() < limite){ return true;}
148         else{
149             Iterator<Pruebas> i = planSemanal.iterator();
150             while(i.hasNext()){
151                 auxiliar = i.next();
```

```
154         if(auxiliar.get_lapsoDias() > 7)
155     {
156         this.add_Tratamiento(auxiliar);
157         i.remove();
158         return true;
159     }
160 }
161 return false;
162 }
163 }
164 }
165
166 /**
167 * busca en la lista el elemento que necesita ser, o bien ejecutado por el enfermero
168 * o bien procesado por el tecnico y lo devuelve. No lo elimina por que esto haria
169 * que la cola menguara y permitiese al trabajador aceptar mas trabajos. Por ello es
170 * requisito indispensable actualizar la cola de plan semanal y comprobar si hay
171 * elementos anteriores a 7 dias antes de meter un elemento en la cola.
172 *
173 */
174 public Pruebas coger_prueba()
175 {
176     if (this.planSemanal.size() > 0)
177     {
178         if(this instanceof Tecnico)
179             for(Pruebas x : planSemanal)
180             {
181                 if(x.get_ejecutado() && !(x.get_procesado())) return x;
182             }
183         if(this instanceof Enfermero)
184         {
185             for(Pruebas x: planSemanal)
186             {
187                 if(!x.get_ejecutado()) return x;
188             }
189         }
190     }
191     return null;
192 }
193
194 /**
195 * comprueba si la cola de trabajo semanal esta vacia
196 */
197 public boolean tiene_Trabajo()
198 {
199     if (this.planSemanal.size() > 0)
200     {
201         if(this instanceof Tecnico)
202             for(Pruebas x : planSemanal)
203             {
204                 if(x.get_ejecutado() && !(x.get_procesado())) return true;
205             }
206         if(this instanceof Enfermero)
207         {
208             for(Pruebas x: planSemanal)
209             {
210                 if(!x.get_ejecutado()) return true;
211             }
212         }
213     }
214     return false;
215 }
216
217 /**
218 * agrega una prueba al plan semanal
219 * @param la prueba a agregar
220 */
221 public void agregar_planSemanal(Pruebas x)
222 {
223     planSemanal.add(x);
224 }
225
226 /**
227 * envia una senhal de alerta a la clinica, comprueba por tiempo
228 * de ejecucion que tipo de trabajador esta dando la alerta y
229 * elije la senhal de acuerdo a sus limitaciones.
230 * alertaVacuna puede mandar 2 senhales, en esta version
231 * solo esta habilitada un ti
```

```
232 *  
233 * @param el trabajador que da la senhal  
234 * @param la prueba razon de la alarma  
235 *  
236 *  
237 *  
238 */  
239 public void alerta_Clinica(Trabajador x, Tratamiento y)  
240 {  
241     if(x instanceof Tecnico && y instanceof Pruebas)  
242         this.clini.alerta((Tecnico)x,(Pruebas)y);  
243     else  
244     {  
245         if((x instanceof Enfermero) && (y instanceof Vacunas))  
246         {  
247             this.clini.alertaVacuna(x,y, false);  
248         }  
249     }  
250 }  
251 }  
252 }  
253 }
```