UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

PROYECTO DE PRÁCTICAS DE PROGRAMACIÓN II
“LA COMUNIDAD DE VECINOS”

Desiderio Almansa Porrero

Asignatura: Fundamentos de Programación II

Titulación: Grado en Ingeniería Informática

Fecha: 02/05/2019

Índice

1. INTRODUCCIÓN: ANÁLISIS DE REQUISITOS.

Debemos implementar un código que nos permita leer una serie de datos que se encuentran en dos ficheros, uno de Vecinos y otro de Sugerencias, y con estos datos realizar una serie de operaciones. A continuación, se indicarán las especificaciones de dicho programa:

Tenemos una comunidad de vecinos de una urbanización.

Los vecinos pueden ser propietarios o inquilinos de un inmueble. De cada uno de ellos sabemos su nombre y apellidos, DNI, inmueble que habita, y número de teléfono de contacto. Entre los inquilinos distinguimos los que alquilan un inmueble durante un año, y aquellos que lo hacen por un mes. En ambos casos se pueden prorrogar, pero no entraremos en ello. Siempre tenemos un inquilino por inmueble. Los alquileres se entienden por meses naturales completos. De cada uno de los inquilinos se tiene el importe del alquiler bien anual, bien mensual según el caso. De entre los inquilinos anuales, tenemos un grupo de inquilinos distinguido, y que etiquetamos como inquilino VIP. A todo inquilino VIP se le ha proporcionado una tarjeta VIP con un número que la identifica.

De cada uno de los propietarios sabemos la fecha de adquisición del inmueble y, además, la cuota anual a pagar a la comunidad por gastos de mantenimiento/servicio. Obsérvese que tal cuota no tiene porqué ser la misma para todos, puesto que no todos los inmuebles son iguales.

Todo miembro de la comunidad puede presentar hasta diez sugerencias en el periodo, de cada una de las cuales se tendrá su tema, descripción y una clasificación en urgente, no urgente y muy urgente.

Tenemos una piscina a la que todo propietario tiene libre acceso.
Un inquilino mensual si quiere acceder a la piscina tiene que pagar una cuota de 30€ por ese mes. Un inquilino anual si quiere acceder a la piscina tiene que pagar una cuota anual de 200€, pero se le aplica un descuento del 5% sobre el importe anual del alquiler, con un importe mínimo a pagar de 100€, ahora bien, si es un inquilino VIP, pagando 50€ de cuota anual puede acceder a la piscina.

No se admitirá ambigüedad en los tipos. Deben utilizarse los adecuados (usar enumerados, si ha lugar), y justificar cuando se trate de simplificaciones.

Las fechas son una clase, nunca una cadena, aunque lógicamente en la representación externa la hagamos así.

Se desean realizar las siguientes consultas:

a) Mostrar toda la información de los vecinos de una comunidad.
b) Un vecino podrá realizar una sugerencia.
c) Consultar el número de sugerencias que ha realizado un vecino dado.
d) Mostrar toda la información de las sugerencias urgentes que ha realizado un determinado vecino.
e) Mostrar la recaudación total de los ingresos de la comunidad de vecinos, y desglosada entre propietarios e inquilinos.
f) Importe que tiene que pagar un determinado vecino.

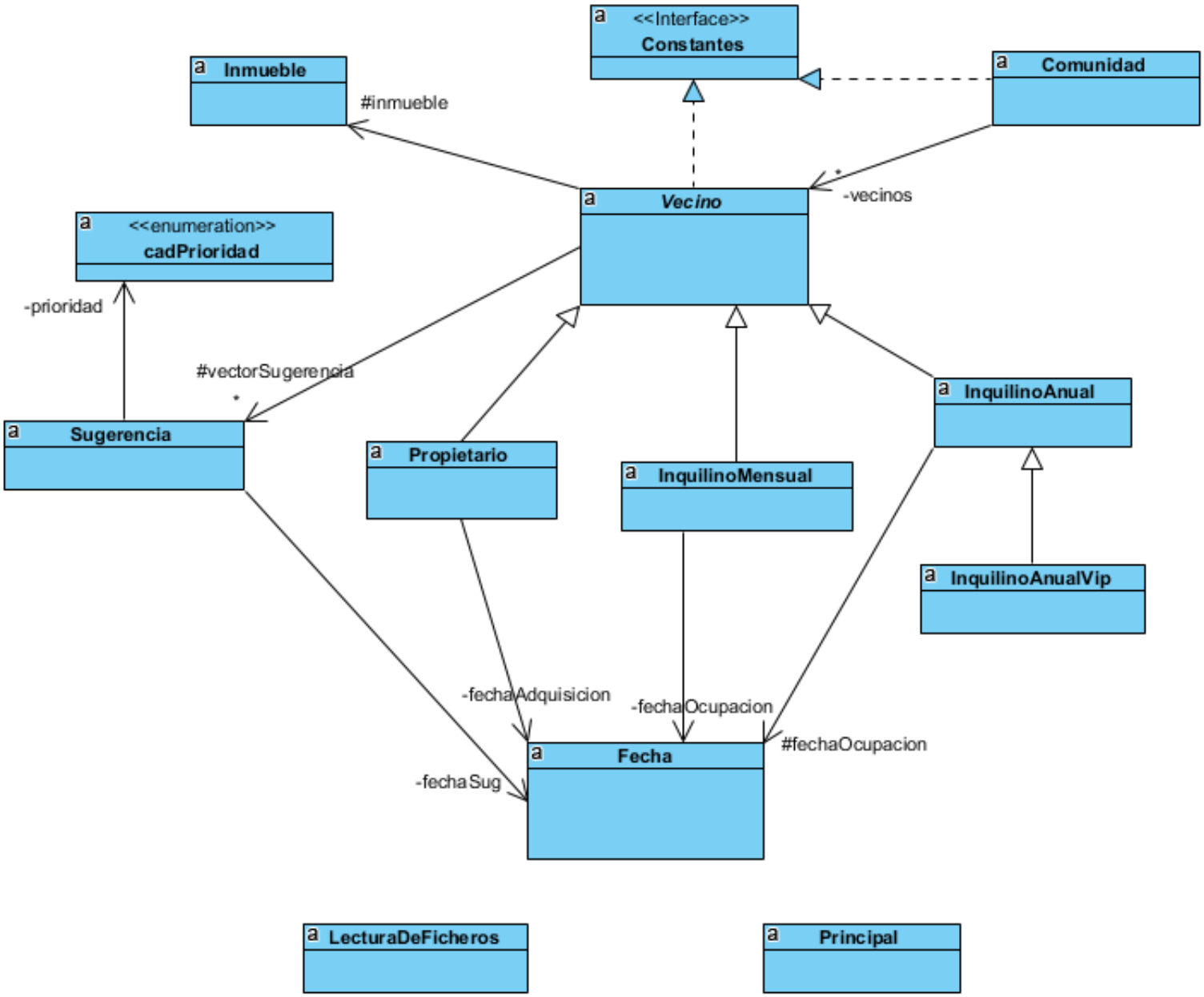
g) Añadir un nuevo vecino a la comunidad.

Extras:

h) Listado por orden alfabético de los vecinos.
i) Listado de inquilinos anuales, ordenado de mayor a menor importe a satisfacer a la comunidad.
j) Listado de propietarios ordenado según fecha de adquisición del inmueble.
k) Listado de los cinco inquilinos que le proporcionan unos mayores ingresos a la comunidad.

2. DISEÑO DEL CÓDIGO.

2.1. RELACIÓN ENTRE CLASES.



En nuestra implementación hemos creado una clase “Vecino” que contiene las características y propiedades comunes a todos los tipos de vecinos de una comunidad. Según los atributos que pueda tener cada vecino se pueden diferenciar entre propietarios, inquilinos mensuales, inquilinos anuales o inquilinos anuales VIP.

Además, como no vamos a instanciar ningún objeto de la clase padre “Vecino”, sino que será siempre de tipo “Propietario”, “InquilinoMensual”, “InquilinoAnual”, “InquilinoAnual” o “InquilinoAnualVip”, hemos considerado conveniente que la clase Vecino sea abstracta.

La clase Vecino tiene un vector de sugerencias como atributo. Como las sugerencias también tienen distintos atributos, como un DNI, una fecha, un asunto, una prioridad y una descripción y todas tienen el mismo comportamiento, se ha creado una clase para ellas, llamada “Sugerencia”.

Para las fechas y los inmuebles, hemos diseñado dos clases: “Fecha” e “Inmueble”, para que sea más fácil su uso, al tratarse de cadenas String.

La clase central de nuestro diseño es la llamada “Comunidad”, que trabaja con un conjunto de vecinos. Es decir, existe una relación de asociación muchos a 1 entre Comunidad-Vecino. Y lo mismo pasa con Vecino-Sugerencia.

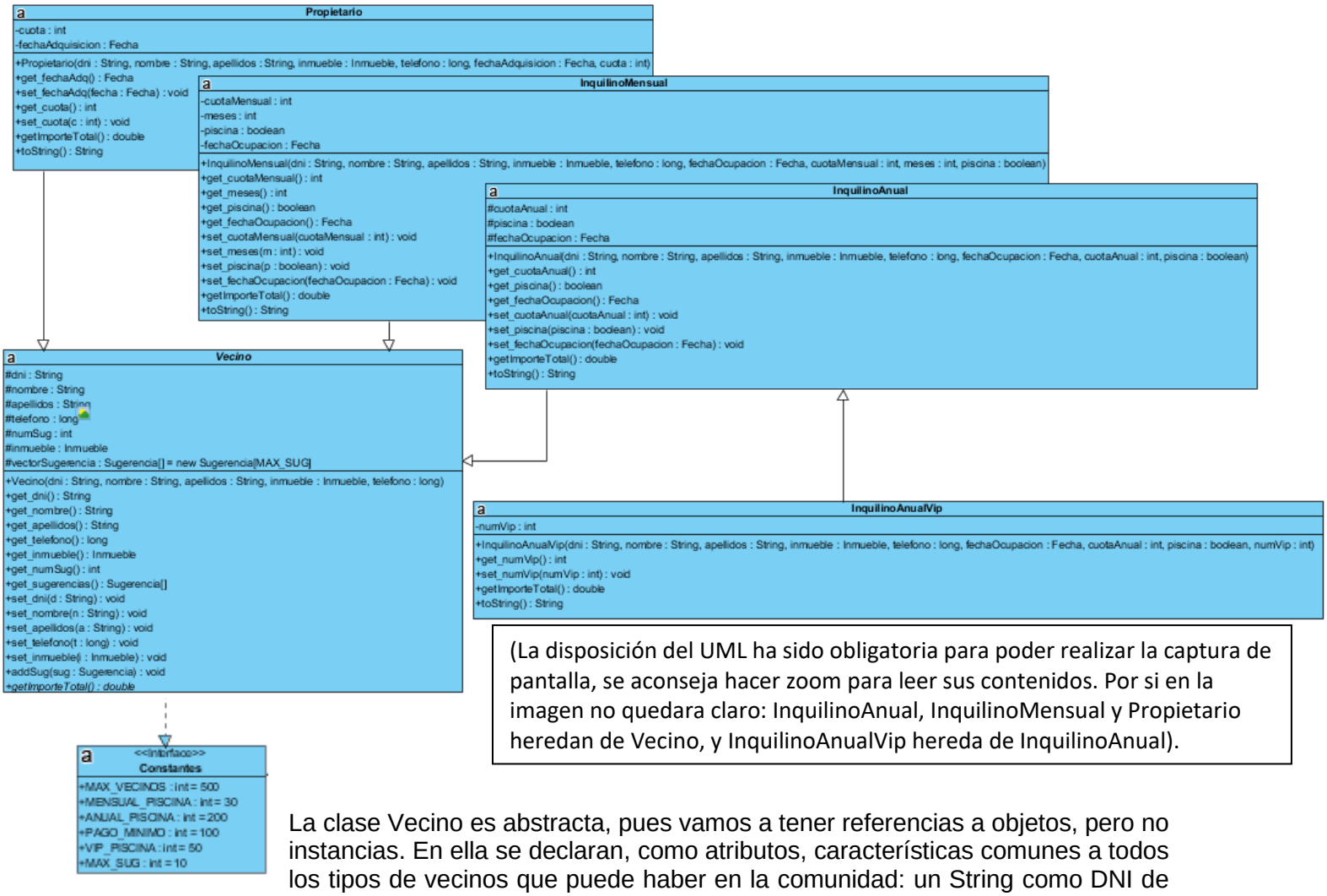
En la clase “LecturaDeFicheros” encontramos dos métodos, uno que lee el fichero que contiene a los vecinos, “leerVecinos(nombreDelFichero, comunidad)”, y otro que lee el fichero que contiene las sugerencias, “leerSugerencias(nombreDelFichero, comunidad)”.

En la clase “Principal” se encuentra el método main, y por tanto es desde la que debemos ejecutar el programa. Esta clase está formada por métodos estáticos que se encargan de imprimir texto en pantalla e invocar métodos de las clases anteriores según el flujo del programa.

Y por último tenemos la interface “Constantes”, en la que se encuentran todas las constantes que usaremos para el programa, y el enum “cadPrioridad”, en la que se encuentran los posibles tipos de prioridad que una sugerencia puede tener.

2.2. DISEÑO DE LAS CLASES.

2.2.1. Vecino, Propietario, InquilinoMensual, InquilinoAnual e InquilinoAnualVip.



(La disposición del UML ha sido obligatoria para poder realizar la captura de pantalla, se aconseja hacer zoom para leer sus contenidos. Por si en la imagen no quedara claro: InquilinoAnual, InquilinoMensual y Propietario heredan de Vecino, y InquilinoAnualVip hereda de InquilinoAnual).

La clase Vecino es abstracta, pues vamos a tener referencias a objetos, pero no instancias. En ella se declaran, como atributos, características comunes a todos los tipos de vecinos que puede haber en la comunidad: un String como DNI de cada vecino (dni), otros dos como el nombre y apellidos (nombre y apellidos), un long que será el número de teléfono (telefono), un inmueble, un entero que contiene el número de sugerencias realizadas por el vecino (numSug) y un vector de objetos Sugerencia que contendrá dichas sugerencias(vectorSugerencia).

Todas estas características tienen delante el modificador protected para permitir que las clases hijas puedan heredarlas.

Hemos programado un único constructor para la clase Vecino, al que se le pasan como parámetros los siguientes atributos: dni, nombre, apellidos, inmueble y telefono. Los dos atributos que faltan en el constructor no creemos que sean necesarios para la construcción del objeto ya que numSug solo es necesario para llevar la cuenta de las sugerencias realizadas (aunque esta se inicializa a 0 dentro), y con respecto al vectorSugerencia hemos pensado que al crear el objeto, éste no tendrá ninguna sugerencia hecha, por lo que no es necesario.

Como métodos de la clase Vecino hemos declarado 8 métodos de consulta (getters) que simplemente proporcionan el valor de alguno de los atributos del objeto. También hemos declarado 5 métodos de actualización (setters), aunque no se usan en ningún momento.

De los métodos de consulta queremos destacar getImporteTotal(), que es un método abstracto, por lo que en Vecino no se sabe que contendrá, pero en las clases hijas si, es el encargado de calcular el importe que paga cada vecino y devolverlo (double).

También, encontramos un método que es un void, addSug(sugerenia), que se encarga de modificar el vectorSugerencia añadiendo las nuevas sugerencias realizadas por el vecino, y además, incrementa numSug cada vez que se realiza.

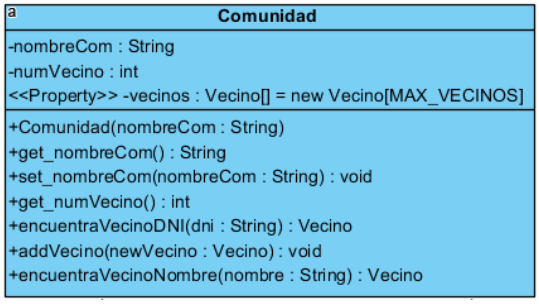
Finalmente, las clases hijas, Propietario, InquilinoMensual e InquilinoAnual, tienen una estructura similar. Cada una tiene un toString() y unos atributos específicos además de los que heredan de la clase padre:

- Propietario: fechaAdquisicion y cuota.
- InquilinoMensual: fechaOcupacion, cuotaMensual, meses y piscina.
- InquilinoAnual: cuotaAnual, fechaOcupacion y piscina.

La clase InquilinoAnualVip hereda, de InquilinAnual, todos los atributos y métodos, que esta ha heredado de Vecino, y también los tres atributos antes mencionados. Esta clase tiene un único atributo propio que es un entero (numVip).

Una última cosa para comentar sobre la clase Vecino es que implementa la interface Constantes.

2.2.2. Comunidad.



Como ya hemos dicho antes, Comunidad es la clase principal, ya que todo el programa gira en torno a ella.
Tiene tres atributos: un String que contiene el nombre de la comunidad (nombreCom), un entero que almacena el número de vecinos que hay en la comunidad (numVecino) y un vector de objetos de tipo Vecino, que contiene todos los vecinos. Este vector está ya inicializado, a MAX_VECINOS. Este dato es el máximo número de vecinos que la comunidad puede tener, y se encuentra en Constantes, por lo que Comunidad implementa esta interface.

A su constructor únicamente le pasamos el atributo nombre, y dentro inicializamos numSug a 0.

En cuanto a los métodos encontramos dos métodos getter (get_nombreCom() y getNumVecino()) y un setter, aunque este no se usa.

También encontramos tres métodos más importantes:

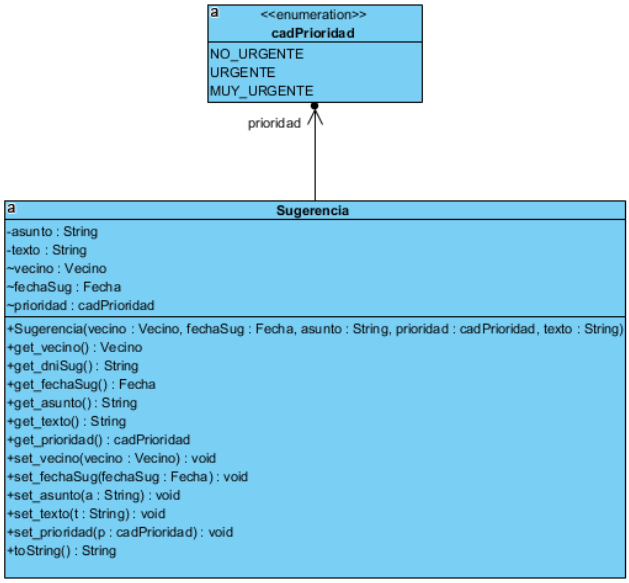
- encuentraVecinoDNI(String dni): Este método es de tipo Vecino, y devuelve un objeto de la clase Vecino. Como parámetro de entrada tiene un String que será el dni del vecino que queramos encontrar. Para ello inicializamos un boolean (encontrado) en

false y un nuevo vecino (v) en nulo. El for, que va desde 0 hasta numVecino y se mantiene activo hasta que el boolean sea cierto. Esto último ocurrirá cuando el DNI del vecino del vector coincida con el DNI introducido como parámetro de entrada, solo entonces el vecino v, que era nulo, se igualará a dicho vecino que tiene el DNI introducido.

- EncuentraVecinoNombre(String nombre): Este método es exactamente igual que el explicado anteriormente, la única diferencia se encuentra en que el parámetro de entrada, y por tanto el que se compara para encontrar al vecino es su nombre, y no su DNI.

- addVecino(Vecino newVecino): Este método es un void, por lo que no devuelve nada, pero realiza una importante operación, añade nuevos vecinos al vector de vecinos de la comunidad (es exactamente igual que addSug(Sugerencia sug) de la clase Vecino). Lo que hace es recibir como parámetro de entrada un objeto de la clase Vecino y, si numVecino es menor que MAX_VECINOS, la posición numVecino del vector vecinos será igual a newVecino (al nuevo vecino). Después de igualarse, numVecinos incrementa en 1.
  Si numVecino es mayor a MAX_VECINOS, no se añadirá el nuevo vecino e imprimirá un mensaje diciendo que no queda espacio para el nuevo vecino.

2.2.3. Sugerencia.

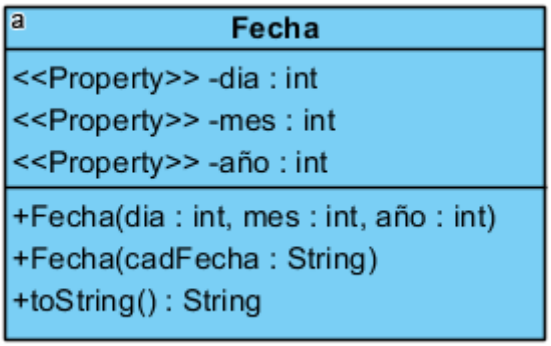


Cada objeto Sugerencia se caracteriza por: un objeto de la clase Vecino (vecino), un objeto de la clase Fecha (fechaSug), un String que contiene el asunto a tratar (asunto), otro String que contiene la descripción de la sugerencia(texto) y un enum de tipo cadPrioridad (prioridad), el cual solo puede ser NO_URGENTE, URGENTE o MUY_URGENTE.

A su constructor le pasamos los atributos o características antes mencionados.

En cuanto a los métodos que contiene, hemos creado un getter y un setter para cada uno de los atributos y, además, uno adicional que devuelve el DNI del vecino al que le pertenece la sugerencia.

2.2.4. Fecha.

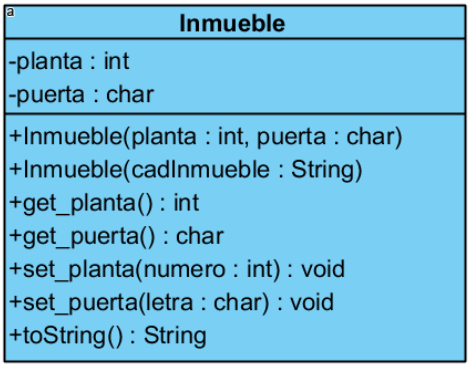


La clase Fecha solamente tiene tres atributos y los tres son enteros. Se corresponden con el día, el mes y el año.

Esta clase se caracteriza por tener dos constructores, uno al que le pasamos los tres enteros mencionados antes (un constructor normal y corriente), que utilizamos en la clase Principal, y otro que utilizamos para la lectura de ficheros. Este último es un poco más complejo, recibe como parámetro un String que lee del fichero Sugerencias.txt (cadFecha) y del que extraeremos los datos día, mes y año, y lo primero que hace es inicializar un StringTokenizer al que le pasamos dicho String (cadFecha) y el "separador" por así decirlo, el carácter que le indica cuando ha de separar el String, que es "/". Luego igualamos los enteros a los fragmentos que va separando (en el mismo orden en el que están escritos dd/mm/aaaa, primero el día, luego el mes y por último el año), pero utilizando la función Integer.parseInt() para transformar los fragmentos separados que son Strings en enteros.

Los métodos que podemos encontrar son tres getters y tres setters para cada atributo y un toString() para imprimir la fecha.

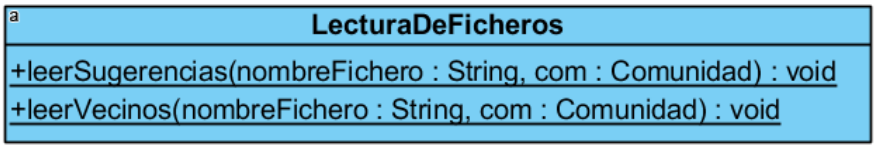
2.2.5. Inmueble.



La clase inmueble tiene dos atributos: un entero que contiene la planta (planta) y un char que contiene la letra de la puerta (puerta). Al igual que la clase Fecha tiene dos constructores, uno normal, al que le pasamos los dos atributos, y otro para la lectura de ficheros. El de la lectura de ficheros recibe como parámetro un String que contiene la cadena leída del fichero Vecinos.txt (cadInmueble) del que extraeremos los dos datos (planta y puerta), después se crea un String que contendrá la planta (cadPlanta) y lo igualamos a todas las posiciones de la cadena de cadInmueble excepto la última que corresponde con la letra. Después la planta la igualamos a cadPlanta, pero transformándola en entero con Integer.parseInt(). Y por último la puerta la igualamos a cadInmueble.charAt(cadInmueble.length()-1) o, en otras palabras, a la última posición de la cadena cadInmueble que contiene únicamente la letra.

Con respecto a los métodos, tiene un getter y un setter para cada atributo, y un toString().

2.2.6. LecturaDeFicheros.



Esta clase no tiene atributos, ya que no se corresponde con un objeto. Contiene dos métodos void que utilizaremos para leer los archivos que necesitamos para el programa, uno para leer el fichero de vecinos y otro para leer el de sugerencias.

- LeerSugerencias(nombreFichero, com): En este método inicializamos un Vecino (vecino), una Fecha (fecha), una cadPrioridad (prioridad) que inicializamos en nulo y cuatro String: uno para el DNI (dni), otro para el asunto (asunto), otro para la descripción (texto) y otro para almacenar cadena con la que luego obtendremos la prioridad de la sugerencia (prioridadLeer).
  Este método contiene dos try-catch, uno dentro de otro. En el primer try inicializamos el FileReader (fr) al que le pasamos el nombre del fichero a leer (nombreFichero), el BufferedReader (br) al que le pasamos el fr, y un String (línea) que contendrá una línea completa del fichero. Después comienza el bucle while que estará ejecutándose siempre que la variable linea no sea nula.
  Dentro de ese while encontramos el otro try en el que está el StringTokenizer (st) al que le pasamos la linea, y que se encargará de separarla en los diferentes tokens que vaya encontrando.
  El primer dato a separar será el DNI, tras esto igualamos el vecino al método encuentraVecinoDNI(dni) creado en comunidad, para encontrar el vecino al que le corresponda dicho DNI. Después el objeto fecha se inicializa, pasándole a su constructor la siguiente cadena o el siguiente token separado. El siguiente token a separar es el asunto, seguido de la prioridad. Para leer la prioridad lo que hemos hecho ha sido almacenar la cadena extraída en la variable prioridadLeer, y después hemos hecho una serie de if-else if en la que miramos si prioridadLeer coincide con “No_Urgente”, “Urgente” o “Muy_Urgente”. Si coincide entonces el cadPrioridad prioridad será igual a una de los tres tipos que puede ser, y si no coincide, entonces la prioridad será nula.
  El último dato a leer es la descripción, que se almacenará en la variable texto.
  Por último se crea un objeto Sugerencia (sug) al que le pasamos todos los datos que hemos ido extrayendo y almacenando en cada variable, y tras esto utilizamos el método, de la clase Vecino, addSug(sug) al que le pasamos la Sugerencia anteriormente creada para que se la añada al Vecino creado al principio.
  Finaliza el try, y comienza el catch de este try, que solamente actúa cuando uno de los datos no es almacenado correctamente en el tipo de variable que es, cuando esto ocurre salta el mensaje “Excepcion leyendo fichero” + nombreFichero.
  Tras esto finaliza el bucle while, y cerramos el FileReader con la función close().
  Se cierra la llave del primer try, y comienza el catch de este try que saltará cuando la línea a leer esté mal escrita y saltará el mismo mensaje que el del otro cathc.

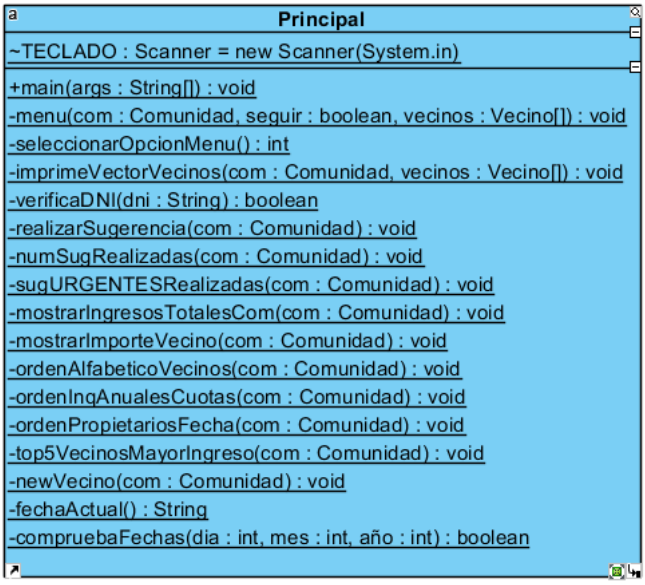
- leerVecinos(nombreFichero, com): Este método es exactamente igual que el anterior. Tiene los mismos try-catch y el bucle while. Lo único que cambia son las variables que este utiliza y el contenido del bucle while.
  Inicializamos un char (tipo) que se corresponde con qué tipo de inquilino es el que vamos a leer del fichero, tres String (dni, nombre y apellidos), un objeto Inmueble (inmueble), un long (teléfono), un objeto Fecha (fecha), un entero (cuota) y un boolean (piscina).
  Dentro del while, y del try, inicializamos el StringTokenizer (st) para que vaya separando los tokens, y vamos leyendo los datos en este orden: tipo, dni, nombre, apellidos, inmueble (le pasamos al constructor la cadena), telefono, fecha (le pasamos al constructor la cadena) y cuota.
  Después de esto hemos creado un switch, que actúa teniendo en cuenta la variable tipo:
  - Si tipo es una 'p', fecha pasa a ser fechaAdquisición, y se crea un objeto de la clase Propietario, que después añadiremos a la comunidad con el método addVecino(com).

  - Si tipo es una 'm', fecha pasa a ser fechaOcupacionM, se crea un entero para los meses que guardará el token que hay después del de la cuota, y también se guarda el token siguiente a este en la variable piscina. Tras esto se crea un objeto de la clase InquilinoMensual que se añade a la comunidad con el método addVecino(com).

  - Si tipo es una 'i', fecha pasa a ser fechaOcupacionA, y se almacena el siguiente token en piscina (el que va después de cuota). Tras esto miramos si el StringTokenizer tiene más tokens para separar de la linea con un if-else. Si hay mas tokens entonces se crea una variable de tipo entero que almacenara dicho token extra (vip) y se crea un objeto de la clase InquilinoAnualVip, que posteriormente se añade a la comunidad. Si no hay mas tokens, se crea un objeto de la clase InquilinoAnual, que posteriormente se añade a la comunidad.

  (Con respecto a lo demás es exactamente igual que leerSugerencias()).

2.3. CLASE PRINCIPAL.



En esta clase es la que se encuentra el método main. Es la única que puede tener algún tipo de interacción con el usuario y que puede mostrar datos por pantalla.

La interacción con el usuario se realiza a través de los distintos métodos que posee, y que se encargan de realizar las operaciones y mostrar por pantalla los textos correspondientes para el uso del programa.

- menu(): Es un void. En este método hacemos el llamamiento de todos los demás. Contiene el switch que se estará ejecutando todo el tiempo, hasta que el usuario marque la opción de salir.

- seleccionarOpcionMenú(): Es de tipo entero. Se encarga de mostrar en pantalla las opciones que el usuario puede elegir, y de devolver dicha elección que se elegirá a través del teclado.

- imprimeVectorVecinos(): Es un void que contiene un bucle for, desde 0 hasta numVecino, que imprime la información de los vecinos.

- verificaDNI(String dni): Es un método de tipo boolean. Tiene como parámetro de entrada un String, que será un DNI. Se encarga de comprobar que la sintáxis de dicho DNI es correcta. Para ello hemos inicializamos un boolean (seguir) a verdadero y luego hemos realizado una serie de if-else. El primer if comprueba que la longitud de la variable dni sea 9, si lo es pasa al siguiente if, que comprueba si la última posición de dni no es una letra, si no lo es, seguir será falso (con la función Character.isLetter()). Si la última posición corresponde con una letra, entonces se pasa a un bucle for, desde 0 hasta 7, que comprueba que las posiciones de la 0 a la 7 son números (con la función Character.isDigit()).
  Si se encuentra una letra, seguir será falso. Si la longitud de dni es 9, la última posición es una letra y el resto números, seguir permanecerá en verdadero.
  El método devuelve el boolean seguir.

- realizarSugerencia(Comunidad com): Es un void. Se encarga de pedir al usuario los datos necesarios para realizar una nueva sugerencia. Es muy parecido al método de la clase LecturaDeFicheros leerSugerencias(), lo que pasa que en este, los datos no se leen desde el fichero y no es necesario separarlos con el StringTokenizer, simplemente se piden al usuario que los vaya introduciendo con el teclado.

- numSugRealizadas(Comunidad com): Se trata de un void. Recibe un objeto de tipo Comunidad como parámetro de entrada. Se encarga de mostrar el número de sugerencias realizadas por un vecino, el cual buscaremos en la comunidad introduciendo su dni en el método encuentraVecinoDNI(dni) de Comunidad.

- sugURGENTESRealizadas(Comunidad com): Es un void. Es muy parecido al método anterior. La diferencia está en que este contiene además un for, que va de 0 a numSug (el número de sugerencias del vecino), y va contando, gracias a un contador (numSugURGENTES), cúantas sugerencias URGENTES se han realizado. Al final en vez de mostrar el número de sugerencias del vecino (numSug), muestra el contador (numSugURGENTES).

- mostrarIngresosTotalesCom(Comunidad com): Se trata de un void al que pasamos una Comunidad como parámetro de entrada. Se encarga de mostrar por pantalla los ingresos generados por cada tipo de vecino y los ingresos totales de la comunidad. Contiene cuatro variables de tipo entero, que actuaran como acumuladores de ingresos. Un for, que va de 0 a numVecino, ira recorriendo el vector de vecinos de la Comunidad, y gracias a unos if-else if, se irá viendo de qué tipo es el vecino. Dependiendo de qué tipo sea se acumularán los ingresos en una variable o en otra (ingresosPropietarios, ingresosInqAnuales o ingresosInqMensuales). En la variable total, se irán acumulando todos los ingresos,

independientemente del tipo que sea cada vecino. Al final devuelve un mensaje con los ingresos generados desglosados en cada tipo de vecino.

- mostrarImporteVecino(Comunidad com): Se trata de un void. Recibe como parámetro de entrada una Comunidad. Se encarga de mostrar el importe total que paga un vecino en concreto, que encontraremos en la comunidad gracias a su dni y al método, de Comunidad, encuentraVecinoDNI(dni).

- ordenAlfabeticoVecinos(Comunidad com): Se trata de un void. Recibe como parámetro de entrada una Comunidad. Se encarga de ordenar los vecinos de la Comunidad alfabéticamente por su nombre. Para ello hemos realizado tres bucles for: el primero, que va de 0 a numVecino, va introduciendo los nombres de cada vecino en un vector de tipo String (nombresVecinos); el segundo contiene otro for anidado, van de 0 a numVecino, y se encargan de ordenar el vector nombresVecinos, para ello es necesario crear una variable auxiliar (variableauxiliar); el último for, que va de 0 a numVecino, se encarga de encontrar a cada vecino gracias al método de Comunidad encuentraVecinoNombre(nombresVecinos[i]) al que le vamos pasando los nombres del vector nombresVecinos, y tras encontrarlos se imprime la información de todos los vecinos (en orden).

- ordenInqAnualCuotas(Comunidad com): Se trata de un void. Recibe como parámetro de entrada una Comunidad. Se encarga de mostrar por pantalla los inquilinos anuales de la comunidad, ordenados según el importe que paguen de mayor a menor. Para ello hemos hecho cuatro bucles for. El primero cuenta cuántos inquilinos anuales hay en la comunidad, y el número se almacena en numInqAnual. Tras esto se crea un vector de objetos InquilinoAnual (inquilinosAnuales) con numInqAnual posiciones. El segundo for va introduciendo los inquilinos anuales de la comunidad en el vector recientemente creado, inquilinosAnuales, y para ello ha sido necesario un contador (cont) que solo incremente cuando se encuentra un inquilino anual. El tercer for, que va de 0 a numInqAnual, tiene otro for anidado, y se encargan de ordenar el vector inquilinosAnuales, comparando los importes de cada uno (para ello es necesario una variable auxiliar, vAuxiliar). El último for se encarga de imprimir la información que se ha solicitado al elegir la opción.

- ordenPropietariosFecha(Comunidad com): Se trata de un void. Recibe como parámetro de entrada una Comunidad. Se encarga de ordenar los propietarios de la comunidad según la fecha en la que compraron el inmueble, de la mas antigua a la mas reciente. Para ello hemos hecho el mismo procedimiento que el anterior método. Solo que en vez de comparar las cuotas, se han comparado las fechas, en el tercer for. Para comparar las fechas se han realizado una serie de if-else if, en los que primero se compara el año, si los años son iguales, se comparan los meses, y si los meses son iguales, se comparan los días, si todos coinciden no se produce ningún cambio en las posiciones.

- top5VecinosMayorIngreso(Comunidad com): Se trata de un void. Recibe como parámetro de entrada una Comunidad. Se encarga de ordenar todos los vecinos de la comunidad según los ingresos que generan, de mayor a menor, y tras esto solo imprime a los cinco primeros, que se corresponden con los que generan unos mayores ingresos. Para ello hemos ordenado el vector igual que en los métodos anteriores (este tipo de ordenamiento se llama “Ordenamiento burbuja”).

- newVecino(Comunidad com): Se trata de un void. Recibe como parámetro de entrada una Comunidad. Se encarga de añadir nuevos vecinos a la comunidad gracias al método de la clase Comunidad, addVecino(). Es muy parecido al método de la clase LecturaDeFicheros, leerVecinos(), se diferencia en que en vez de leer los datos con un FilieReader y separarlos con un StringTokenizer, lo que hace es pedir al usuario que los vaya introduciendo por teclado.

- fechaActual(): Se trata de un método de tipo String, que devuelve un String (fechaString) que contiene la cadena con la fecha que se le pasará al constructor de Fecha, al crear una sugerencia nueva. Al ser la sugerencia nueva debe llevar la fecha actual. Para ello hemos utilizado el SimpleDateFormat (formateador) y un objeto de tipo Date (fechaActual). A formateador le pasamos el formato en el que queremos que esté nuestra cadena que contendrá la fecha. Y fechaActual recoge la fecha actual de tu ordenador. Después se iguala fechaString a fechaActual, pero formateada con el SimpleDateFormat, gracias a la función formateador.format(fechaActual).

- compruebaFechas(int día, int mes, int año): Se trata de un método de tipo boolean, que devuelve un boolean (fechaErronea). Se encarga de comprobar que la fecha introducida por el usuario, cuando se crea un nuevo vecino (cuando le pedimos la fecha en la que compró el inmueble), sea correcta, es decir, que los días, meses y años tengan sentido y concuerden con los años normales (no bisiestos). Para ello hemos hecho una serie de if- else if, que devuelven que fechaErronea es verdadero si:
  - los días son menores que 1 o mayores que 31 para los meses 1, 3, 5, 7,8,10 y 12.
  - los días son menores que 1 o mayores que 30 para los meses 4, 6, 9 y 11.
  - los días son menores que 1 o mayores que 28 para el mes 2.
  - los meses son menores que 1 o mayores que 12.
  - los años son menores que 1900 o mayores que 2019.

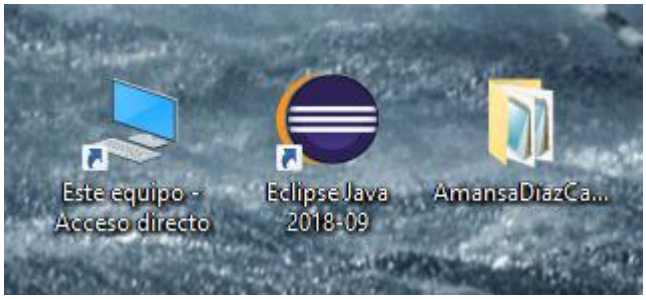
3. MANUAL DE INSTALACIÓN.

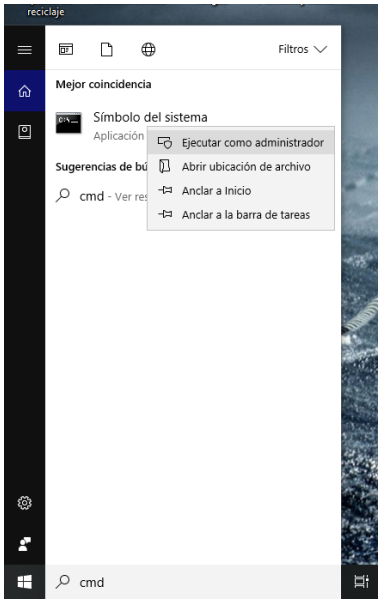
En este apartado se mostrará de manera detallada como compilar y ejecutar nuestro programa desde la consola de comandos.

Nota: Para poder ejecutar archivos java desde la consola de comandos es necesario tener instalado y configurado en nuestro ordenador la última versión de la herramienta JDK, la cual se puede descargar desde la página oficial (www.oracle.com).

PASO 1: Situar el archivo "AlmansaDiazCabello" en un lugar fácilmente accesible:

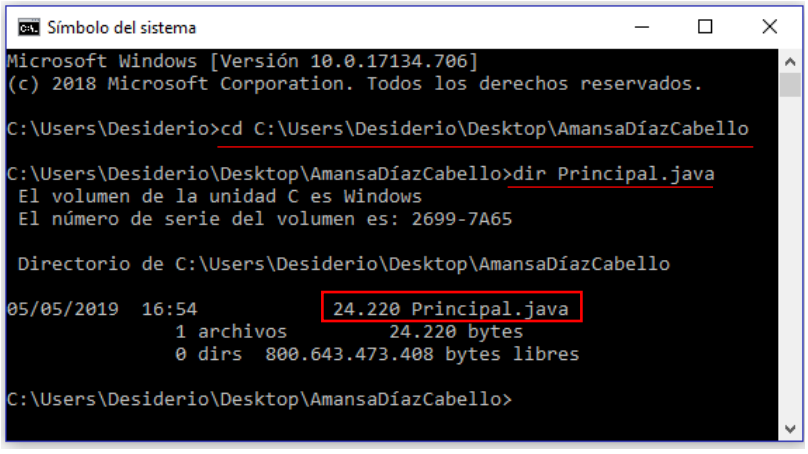
Puede ser el escritorio, la raíz de algún disco local o incluso una memoria extraíble. Es recomendable situarlo en el escritorio, pues así lo aremos en esta explicación.





PASO 2: Abrir la aplicación símbolo del sistema (cmd): Para ello accedemos al buscador de Windows y escribimos "cmd". Haz clic derecho sobre él y selecciona "Ejecutar como administrador".

PASO 3: Localizar el archivo "Principal.java": Para ello, dentro de la consola de comandos tenemos que dirigirnos al directorio en el que este situado nuestro programa (paso 1). Introduciremos "cd" y la ruta de la carpeta "AlmansaDiazCabello" (para situarnos dentro de ella). Tras esto comprobamos que el programa se encuentre ahí, escribiendo "dir Principal.java".



PASO 4: Compilar: Una vez localizado el programa, escribimos la instrucción “javac Principal.java”. Si se ha compilado correctamente debería haberse creado un archivo .clas para cada clase en la misma dirección. Podremos comprobar esto usando el comando “dir”.

PASO 5: Ejecutar: Escribimos la instrucción “java Principal”, y el programa comenzará a ejecutarse.

```
C:\Users\Desiderio>cd C:\Users\Desiderio\Desktop\AmansaDíazCabello

C:\Users\Desiderio\Desktop\AmansaDíazCabello>javac Principal.java

C:\Users\Desiderio\Desktop\AmansaDíazCabello>java Principal
Indique que operación desea realizar:
1. Mostrar informacion de los vecinos.
2. Realizar sugerencia.
3. Consultar número de sugerencias realizadas.
4. Mostrar toda la información de las sugerencias URGENTES que ha realizado un determinado vecino.
5. Mostrar ingresos totales de la comunidad al año (desglosado).
6. Mostrar importe que tiene que pagar un vecino concreto.
7. Listado en orden alfabetico de los vecinos.
8. Listado de inquilinos anuales, ordenado de mayor a menor importe a satisfacer a la comunidad.
9. Listado de propietarios ordenado según la fecha de adquisición del inmueble.
10. Listado de los cinco vecinos que le proporcionan unos mayores ingresos a la comunidad.
11. Añadir un nuevo vecino.
12. Salir.
```

4. MANUAL DE USUARIO.

Al ejecutar el programa, la primera operación que realiza es leer los ficheros Vecinos.txt y Sugerencias.txt, aunque no es necesario para su correcto funcionamiento ya que podremos ir añadiendo vecinos y sugerencias introduciéndolas por teclado.

Aparecerá un texto indicándonos todas las opciones que el programa puede realizar.

```
Indique que operación desea realizar:
1. Mostrar informacion de los vecinos.
2. Realizar sugerencia.
3. Consultar número de sugerencias realizadas.
4. Mostrar toda la información de las sugerencias URGENTES que ha realizado un determinado vecino.
5. Mostrar ingresos totales de la comunidad al año (desglosado).
6. Mostrar importe que tiene que pagar un vecino concreto.
7. Listado en orden alfabetico de los vecinos.
8. Listado de inquilinos anuales, ordenado de mayor a menor importe a satisfacer a la comunidad.
9. Listado de propietarios ordenado según la fecha de adquisición del inmueble.
10. Listado de los cinco vecinos que le proporcionan unos mayores ingresos a la comunidad.
11. Añadir un nuevo vecino.
12. Salir.
```

1.Mostrar información de los vecinos.

Si pulsamos esta opción, simplemente nos mostrará toda la información de todos los vecinos que haya en la comunidad.

2.Realizar sugerencia.

```
Introduzca los siguientes datos:

DNI: 12345678H
Fecha de hoy: 05/05/2019
Asunto: Piscina
Prioridad :
 1.No urgente
 2.Urgente
 3.Muy urgente
2
Descripcion: Demasiado_cloro
```

Si elegimos la opción 2, nos pedirá introducir una serie de datos. Primero el DNI del vecino que quiere realizar la sugerencia, si el DNI está mal escrito se volverá a pedir, pero si escribimos el DNI de alguien que no está en la comunidad saldrá un mensaje y la opción se anulará, y saldrá de nuevo el menú de opciones. Tras introducir correctamente el DNI, nos muestra la fecha de la realización de la sugerencia, que es la actual. Después va el asunto, que con una o dos palabras es suficiente. Luego la prioridad, que elegiremos introduciendo los números como se muestra (1-no urgente, 2-urgente, 3-muy urgente), y si elegimos un número que no sea uno de estos tres, se volverá a pedir. Por último nos pide que introduzcamos la descripción. AVISO: No podemos utilizar espacios, ya que el programa da error, debemos usar barra baja "_".

3.Consultar número de sugerencias realizadas.

```
Introduzca el DNI del vecino del que quiera conocer el numero de sugerencias realizadas:
12345678H

El vecino Hernan Cortes_Monroy ha realizado un total de 1 sugerencias.
```

Nos pide que introduzcamos el DNI del vecino de que queramos saber el número de sugerencias que ha realizado. El procedimiento será el mismo para todas las veces que nos pida el DNI, independientemente de la opción que hallamos elegido, si está mal la sintaxis, nos pedirá que lo introduzcamos de nuevo, si el DNI no se encuentra en la comunidad se anulará la opción y saltara de nuevo al menú. Tras introducirlo nos muestra un mensaje con el número de sugerencias realizadas.

4.Mostrar información de las sugerencias URGENTES que ha realizado un determinado vecino.

```
Introduzca el DNI del vecino del que quiera conocer toda la información sobre las sugerencias URGENTES:
12345678H

Sugerencia nº 1[DNI del vecino: 12345678H, Fecha: 23/3/2018, Asunto: Piscina, Prioridad: URGENTE, Descripcion: Puerta_entrada_rota]

El vecino Hernan Cortes_Monroy ha realizado un total de 1 sugerencias URGENTES.
```

Al igual que el anterior, pide el DNI del vecino y nos muestra la información de las sugerencias URGENTES y el número que ha realizado de ellas.

6.Mostrar importe que tiene que pagar un vecino concreto.

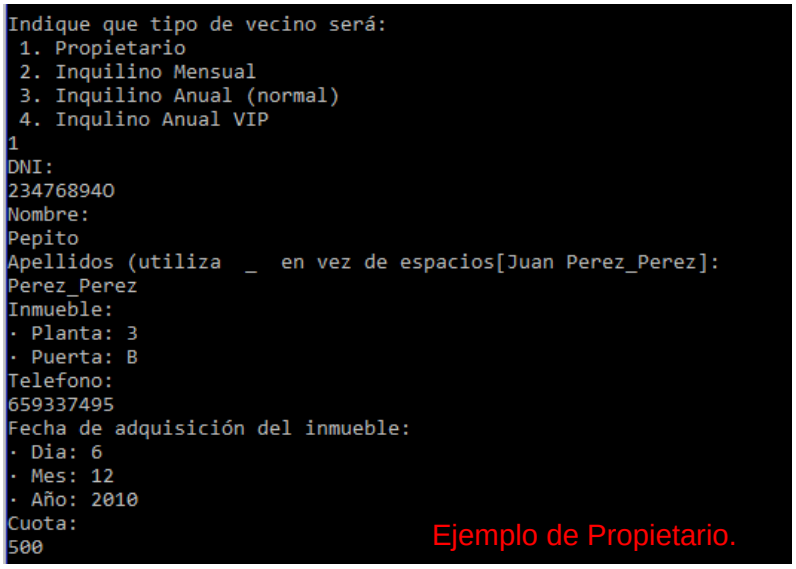
```
Introduzca el DNI del vecino del que quiera conocer el importe:
12345678H
El vecino Hernan Cortes_Monroy paga un importe de 3600.0 euros al año.
```

Exactamente igual que las anteriores: pide el DNI y te muestra la información de dicho vecino.

Opciones 5, 7, 8, 9 y 10.

No creemos que sea necesario hacer un apartado exclusivo para cada una, por eso hemos decidido ponerlas juntas.
Su funcionamiento es el más sencillo de todos, ya que simplemente debemos seleccionarlas para que nos muestre la información. No es necesario que el usuario introduzca nada más.

11.Añadir nuevo vecino.



Ejemplo de Propietario.

Con esta opción podremos añadir nuevos vecinos a la comunidad.
Primero nos pide el tipo de vecino que será (1.Propietario, 2.Inq. Mensual, 3.Inq. Anual o 4.Inq. Anual Vip), y debemos introducir uno de los números que nos indica, dependiendo de qué tipo queremos elegir. Si elegimos otro número que no sea los que nos indica nos volverá a pedir que lo introduzcamos de nuevo. Tras esto nos pedirá el DNI, si está mal escrito lo volverá a pedir, y si escribimos el de uno de los vecinos ya existentes, la opción se cancelara y saldrá al menú. Después nos pedirá nombre y apellidos (los espacios se escriben con "_"). Luego la planta y la puerta, si para la puerta escribimos un número la volverá a pedir. Después pide el número de teléfono y la fecha de adquisición/ocupación del inmueble. La fecha debe introducirse dentro de los parámetros reales, es decir, para febrero solo se pueden introducir los días del 1 al 28, solo hay 12 meses, etc, si no volverá a pedirla por completo. Y después pide los demás datos correspondientes a cada tipo de vecino.

12. Salir.

Si elegimos la última opción, el programa cerrará, por lo que el menú ya no saldrá más hasta que volvamos a ejecutar el programa. (si pulsamos ctrl + c, también se cierra)

5. ESFUERZO REALIZADO.

En horario de clase se han empleado aproximadamente 15 horas.

En casa se han empleado aproximadamente 20 horas (quizá más).

Total, horas empleadas en la realización del código y la memoria: 35 (quizá más).

6. BIBLIOGRAFÍA.

- StringTokenizer - https://docs.oracle.com/javase/7/docs/api/java/util/StringTokenizer.html
- Formas de ordenar un array - https://www.discoduroderoer.es/formas-de-ordenar-un-array-en-java/
- SimpleDateFormat - http://felinfo.blogspot.com/2009/05/fecha-actual-en-espanol-con-java.html
- Transparencias de clase.
- YouTube “pildorasinformáticas” - https://www.youtube.com/user/pildorasinformaticas (Curso de Java desde 0)