```java
/**
 * @ (#) GestionTaller.java
 *
 * Clase GestionTaller.
 * Esta clase va a gestionar el conjunto de objetos de las clases Vehiculo, Mecanico, Cliente, Ficha y Oferta.
 * Por tanto, los atributos de esta clase son:
 * De tipo ArrayList: vehiculos y mecanicos.
 * De tipo HashMap: clientes, fichas y ofertas.
 * Se han elegido dos tipos de colecciones para trabajar diferentes estructuras.
 *
 * @author Yassine Marroun
 * @version 1.00 2017/05/24
 */
import java.util.*;

public class GestionTaller {

    private List<Vehiculo> vehiculos = new ArrayList<Vehiculo>();
    private List<Mecanico> mecanicos = new ArrayList<Mecanico>();
    private Map<String, Cliente> clientes = new HashMap<String, Cliente>();
    private Map<Integer, Ficha>  fichas =  new HashMap<Integer, Ficha> ();
    private Map<String, Oferta>  ofertas =  new HashMap<String, Oferta> ();


    // Hacemos uso del método createDatosInicialesPruebas para añadir una serie de objetos
    // de las clases mencionadas a sus correspondientes listas y mapas.
    // Es de utilidad para realizar pruebas haciendo diferentes selecciones en el menú.

    public void createDatosInicialesPruebas() {

    //  Comercial comercial1 = new Comercial ("Susana Oliver", 625483275, "Comercial Primero.");
    //  Jefe jefe1 = new Jefe ("Victor Machado", 633144895, "Gerente.");
        Mecanico mecanico1 = new Mecanico ("Jorge Garrido", 645786329, "Primer Mecanico.");
        Mecanico mecanico2 = new Mecanico ("Carlos Bueno", 685147956, "Segundo Mecanico.");
        Mecanico mecanico3 = new Mecanico ("Sergio Ibaiz", 654359624, "Tercer Mecanico.");
        mecanicos.add(mecanico1);
        mecanicos.add(mecanico2);
        mecanicos.add(mecanico3);

        Cliente cliente1 = new Cliente("Pedro", 632012587, "2453175S", "pedro@tal");
        Cliente cliente2 = new Cliente("Isabel", 693214758, "3785425A", "isa@tal");
        Cliente cliente3 = new Cliente("Daniel", 630247852, "41236984H", "dani@tal");
        Cliente cliente4 = new Cliente("Juan", 916325748, "75216235P", "juan@tal");

        Coche vehiculo1 = new Coche(Enumerados.TipoCoche.TURISMO, "Opel", "Astra", "1128DFT", Enumerados.Combustible.DIESEL, "2453175S");
        Coche vehiculo2 = new Coche(Enumerados.TipoCoche.MONOVOLUMEN, "Seat", "Toledo", "3378ERD", Enumerados.Combustible.DIESEL, "2453175S");
        VehiculoGrande vehiculo3 = new VehiculoGrande(Enumerados.TipoGrandes.FURGONETA, "BMW", "X5", "8756FTP", Enumerados.Combustible.GASOLINA, "41236984H");
        Moto vehiculo4 = new Moto(Enumerados.TipoMoto.DE_CARRETERA, "Kawassaki", "500", "9620GEA", Enumerados.Combustible.GASOLINA, "3785425A");
        Coche vehiculo5 = new Coche(Enumerados.TipoCoche.VEHICULO_PROFESIONAL, "Ambulancia", "PequeÃ±a", "4875DER", Enumerados.Combustible.GASOLINA, "3785425A");

        vehiculos.add(vehiculo1);
        vehiculos.add(vehiculo2);
        vehiculos.add(vehiculo3);
        vehiculos.add(vehiculo4);
        vehiculos.add(vehiculo5);
```

```
56
57          cliente1.addVehiculo(vehiculo1);
58          cliente1.addVehiculo(vehiculo2);
59          cliente3.addVehiculo(vehiculo3);
60          cliente2.addVehiculo(vehiculo4);
61          cliente2.addVehiculo(vehiculo5);
62
63          clientes.put("2453175S", cliente1);
64          clientes.put("3785425A", cliente2);
65          clientes.put("41236984H", cliente3);
66          clientes.put("75216235P", cliente4);
67
68          ArrayList<Trabajo> trabs = new ArrayList<Trabajo>();
69          trabs.add(new Trabajo(Enumerados.MotivoVisita.MOTOR));
70          Ficha ficha1 = new Ficha(1,"8756FTP", "Jorge Garrido", trabs);
71
72          fichas.put(1, ficha1);
73
74          ArrayList<Oferta> oferts = new ArrayList<Oferta>();
75          Oferta oferta1 = new Oferta("ofe1", Enumerados.TipoOferta.CAMBIO_DE_ACEITE, 15);
76          oferts.add(oferta1);
77          cliente1.setOfertas(oferts);
78          cliente2.setOfertas(oferts);
79
80          ofertas.put("ofe1", oferta1);
81      }
82
83
84      /*
85       Métodos para gestionar objetos de la clase Vehiculo.
86       */
87
88      // El método crearVehiculo crea el objeto vehiculo, le asigna los datos obtenidos con
    el método nuevoVehiculo,
89      // lo añade al ArrayList vehiculos y, por último, con el dni localiza al cliente en el
    mapa de clientes
90      // y con addVehiculo lo asigna a la lista de vehículos de dicho cliente.
91
92      public void crearVehiculo(){
93            Vehiculo vehiculo = new Vehiculo();
94            vehiculo = vehiculo.nuevoVehiculo();
95            if (vehiculo!=null){
96                vehiculos.add(vehiculo);
97                Cliente cl = clientes.get(vehiculo.getDniCliente());
98                if (cl!=null){
99                  cl.addVehiculo(vehiculo);
100               }
101           } else{
102               System.out.print("Fallo dando de alta un vehiculo");
103           }
104     }
105
106
107     // El método buscarVehiculoPantalla nos pide que le demos por teclado una matricula,
108     // se la pasa como parámetro a buscarVehiculo e imprime en pantalla los datos del
    vehiculo que le devuelve.
109
110     public void buscarVehiculoPantalla(){
111         String matricula;
112         System.out.print("Buscar Vehiculo por Matricula: ");
113         matricula = Menu.sc.nextLine();
114         Vehiculo vehiculo = buscarVehiculo(matricula);
```

```
115         if (vehiculo!=null){
116             System.out.println(vehiculo.toString());
117         } else{
118             System.out.println("vehiculo no existe");
119         }
120     }
121 
122 
123     // El método buscarVehiculo recibe una variable matricula, recorre el ArrayList vehiculos
124     // y devuelve el objeto vehiculo que localice con dicha matricula.
125 
126     public Vehiculo  buscarVehiculo(String matricula){
127         for(int i = 0; i < vehiculos.size(); i++){
128             if(vehiculos.get(i).getMatricula().equals(matricula)){
129                 return vehiculos.get(i);
130             }
131 
132         }
133         return null;
134     }
135 
136 
137     // El método eliminarVehiculo nos pide una matricula, hace uso de dicha variable
138     // para recorrer el ArrayList vehiculos y elimina el objeto vehiculo que localice con dicha matricula.
139 
140     public void eliminarVehiculo() {
141         Scanner sc = new Scanner(System.in);
142         String matricula;
143         System.out.print("Eliminar Vehiculo por Matricula: ");
144         matricula = sc.nextLine();
145 
146         for(int i = 0; i < vehiculos.size(); i++) {
147             if (vehiculos.get(i).getMatricula().equals(matricula)) {
148                 vehiculos.remove(i);
149             }
150         }
151         sc.close();
152     }
153 
154 
155     // El método listarVehiculos muestra en pantalla todos los elementos que tengamos en el ArrayList vehiculos.
156 
157     public void listarVehiculos() {
158         System.out.println("Listado de vehiculos.");
159         for(Vehiculo vhi: vehiculos) {
160             System.out.println(vhi.toStringAmpliado());
161         }
162     }
163 
164 
165     /*
166      Métodos para gestionar objetos de la clase Cliente.
167      */
168 
169     // El método crearCliente crea un objeto cliente, le asigna los datos obtenidos mediante
170     // el método nuevoCliente y se lo pasa como parámetro a guardarCliente,
171     // que es donde se va a incluir en el mapa de clientes.
172 
```

```
173     public void crearCliente(){
174
175         Cliente cliente = new Cliente();
176         cliente = cliente.nuevoCliente();
177         guardarCliente(cliente);
178
179     }
180
181
182     // El método buscarClientePantalla nos pide que le demos por teclado un dni,
183     // se lo pasa como parámetro a buscarCliente e imprime en pantalla el objeto cliente
    que le devuelve.
184
185     public void buscarClientePantalla(){
186         String dni;
187         System.out.print("Buscar Cliente por dni: ");
188         dni = Menu.sc.nextLine();
189         Cliente cliente = buscarCliente(dni);
190         if (cliente!=null){
191             System.out.println(cliente.toString());
192         } else{
193             System.out.println("cliente no existe");
194         }
195     }
196
197
198     // El método buscarCliente devuelve del HashMap clientes el objeto cliente con el que
    coincida
199     // el dni que se le ha pasado como parámetro.
200
201     public Cliente buscarCliente(String dni){
202         return clientes.get(dni);
203     }
204
205
206     // El método guardarCliente hace uso del dni para comprobar si existe el objeto
    cliente,
207     // si ya existe nos da un mensaje de error y si no es así, lo añade al HashMap
    clientes.
208
209     public void guardarCliente(Cliente cliente){
210         Cliente cl = clientes.get(cliente.getDni());
211         if (cl != null) {
212             System.out.println("No se puede introducir el Cliente. Ya esta registrado.");
213         } else {
214             clientes.put(cliente.getDni(), cliente);
215         }
216     }
217
218
219     // El método eliminarCliente elimina el objeto cliente con el que coincida
220     // el dni que se le ha pasado como parámetro.
221
222     public void eliminarCliente(String dni) {
223         clientes.remove(dni);
224     }
225
226
227     // El método listarClientes simplemente muestra en pantalla los elementos que tengamos
    en el HashMap clientes.
228
229     public void listarClientes(){
```

```
230         for (Cliente cl : clientes.values()) {
231             System.out.println(cl.toString());
232         }
233     }
234 
235 
236     /*
237      Métodos para gestionar objetos de la clase Ficha.
238      */
239 
240     // El método getNumNuevaFicha obtiene el total de fichas que tenemos en el HashMap
    fichas y le suma uno.
241 
242     public int getNumNuevaFicha() {
243         if (fichas!=null && fichas.size()>0){
244            return fichas.size() + 1;
245         }else{
246            return 1;
247         }
248     }
249 
250 
251     // El método altaFichas crea una nueva ficha con el número recibido en el parámetro
    nuevoNumero
252     // y los datos obtenidos por el método nuevaFicha.
253 
254     public void altaFichas(Integer nuevoNumero){
255           Ficha ficha = new Ficha();
256           ficha = ficha.nuevaFicha(nuevoNumero);
257           if (ficha!=null){
258               guardarDatosFicha(nuevoNumero, ficha);
259           }
260     }
261 
262 
263     // modificarFicha recibe un número de ficha, si no existe ficha con dicho número da un
    mensaje de error,
264     // y si existe, muestra los datos de la ficha a modificar y solicita los nuevos datos
    de trabajos a realizar,
265     // estado en el que se encuentra y si está Terminado, solicita la fecha. Con esos datos
    guarda
266     // una nueva ficha que sustituye la anterior en el mapa de fichas con el mismo número
    de ficha.
267 
268     public void modificarFicha(Integer numFic){
269        Ficha fichaAnt = fichas.get(numFic);
270        Ficha fichaNueva = null;
271        if (fichaAnt!=null){
272             fichaNueva = fichaAnt.modificarFicha(fichaAnt);
273        } else{
274             System.out.println("Numero de ficha inexistente");
275        }
276        if (fichaNueva!=null){
277            guardarDatosFicha(numFic, fichaNueva);
278        }
279     }
280 
281 
282     // El método guardarDatosFicha va a comprobar si por las características del vehiculo
283     // hay que asignarle algún trabajo más. Por ejemplo, si en su atributo de tipo
    enumerado Combustible
284     // hemos guardado que es un vehículo diésel, se le añade la revisión del filtro de
```

```
   partículas.
285     // Tras los cambios añade la ficha al mapa y la imprime en pantalla.
286
287     private void guardarDatosFicha(Integer numFic, Ficha ficha){
288           String matricula = ficha.getMatricula();
289           Vehiculo vehi = buscarVehiculo(matricula);
290           if (vehi!=null){
291               Trabajo trab;
292               if (vehi instanceof Moto){
293                   trab = new Trabajo(Enumerados.MotivoVisita.PRESION_NEUMATICOS);
294                   ficha.getTrabajos().add(trab);
295               }
296               if (vehi instanceof Coche){
297                   Coche coche = (Coche) vehi;
298                   if
   (coche.getTipoCoche().equals(Enumerados.TipoCoche.VEHICULO_PROFESIONAL)){
299                       trab = new Trabajo(Enumerados.MotivoVisita.SIRENAS);
300                       ficha.getTrabajos().add(trab);
301                   }
302               }
303               if (vehi.getCombustible().equals(Enumerados.Combustible.DIESEL)){
304                   trab = new Trabajo(Enumerados.MotivoVisita.FILTRO_PARTICULAS);
305                   ficha.getTrabajos().add(trab);
306               }
307               fichas.put(numFic, ficha);
308               System.out.println(ficha.toStringReducido());
309           } else{
310               System.out.println("La matricula grabada en la ficha no existe. Dar de alta
   el vehiculo");
311           }
312     }
313
314
315     // El método visualizaFichasTodas simplemente imprime en pantalla todos los elementos
316     // que tenemos en el HashMap fichas.
317
318     public void visualizaFichasTodas() {
319         System.out.println("Listado de fichas.");
320         for(Ficha fic: fichas.values()) {
321             System.out.println(fic.toStringReducido());
322         }
323     }
324
325
326     // El método visualizaFichasMecanico recorre el mapa de fichas comparando el nombre del
   mecánico
327     // de cada ficha con el nombre del mecánico pasado en el parámetro, en el caso de
   coincidir,
328     // imprime esa ficha. Cada vez que coincide el nombre, imprime esa ficha.
329
330     public void visualizaFichasMecanico(String nomMecanico) {
331         boolean existe = false;
332         System.out.println("Listado de fichas del mecanico: " + nomMecanico);
333         for(Ficha fic: fichas.values()) {
334             if (fic.getNombreMecanico().equals(nomMecanico)){
335                 System.out.println(fic.toString());
336                 existe = true;
337             }
338         }
339         if (!existe){
340             System.out.println("No existen fichas de este mecanico");
341         }
```

```
342     }
343
344
345     // visualizaFichasEstado realiza una búsqueda en el HashMap fichas e imprime en
    pantalla
346     // aquellas en las que coincide su estado con el que le hemos pasado como parámetro al
    método.
347     // En el enunciado se pedía que se mostrasen las fichas que se encuentren en estado
    Parado,
348     // pero con este método podemos mostrar las fichas que se encuentre en el estado que
    seleccionemos.
349
350     public void visualizaFichasEstado(Enumerados.EstadoReparacion estado) {
351         boolean existe = false;
352         System.out.println("Listado de fichas en estado: " + estado.toString());
353         for(Ficha fic: fichas.values()) {
354             if (fic.getEstado().equals(estado)){
355                 System.out.println(fic.toStringReducido());
356                 existe = true;
357             }
358         }
359         if (!existe){
360             System.out.println("No existen fichas en este estado");
361         }
362     }
363
364
365     // El método fichasEntreFechas va a solicitar dos fechas mediante el método
    darFechaConsola,
366     // y recorriendo el mapa de fichas imprime en pantalla todas aquellas fichas en las que
367     // el método entrefechas es true. Si no existe ninguna, daría el mensaje de que no
    existen fichas.
368
369     public void fichasEntreFechas() {
370         Ficha ficFec = new Ficha();
371         Calendar fecha1 = ficFec.darFechaConsola();
372         Calendar fecha2 = ficFec.darFechaConsola();
373         boolean existe = false;
374         if (fecha1!=null && fecha2!=null){
375             System.out.println("Listado de fichas reparadas entre fechas: " +
    fecha1.getTime() + " y " + fecha2.getTime());
376             for(Ficha fic: fichas.values()) {
377                 if (fic.entreFechas(fic.getFechaReparacion(), fecha1, fecha2)){
378                     System.out.println(fic.toStringReducido());
379                     existe = true;
380                 }
381             }
382         }
383         if (!existe){
384             System.out.println("No existen fichas reparadas entre esas fechas");
385         }
386     }
387
388
389     // El método asignarFichaAMecanico recibe como parámetros las variables numFicha y
    nombreMecanico.
390     // Si el método buscarMecanico le devuelve que existe un objeto mecanico con ese mismo
    nombreMecanico,
391     // le asigna ese dato al atributo nombreMecanico de la ficha indicada por la variable
    numFicha.
392
393     public void asignarFichaAMecanico(Integer numFicha, String nombreMecanico){
```

```
394         Ficha ficha = fichas.get(numFicha);
395         Boolean existe = false;
396         if (ficha!=null){
397             existe = buscarMecanico(nombreMecanico);
398             if (existe){
399                 ficha.setNombreMecanico(nombreMecanico);
400             } else{
401                 System.out.println("No existe el mecanico: " + nombreMecanico);
402             }
403         } else {
404             System.out.println("No existe la ficha: " + numFicha);
405         }
406     }
407 
408 
409     // El método getFicha recibe un numFicha y devuelve el objeto ficha con el que coincida del HashMap fichas.
410 
411     public Ficha getFicha(Integer numFicha){
412         return fichas.get(numFicha);
413     }
414 
415 
416     /*
417      Método para gestionar objetos de la clase Usuario.
418      */
419 
420     // El método buscarMecanico es de tipo boolean y únicamente devuelve true o false si localiza o no
421     // un objeto mecanico mediante la variable nombreMecanico que le hemos pasado.
422 
423     private Boolean buscarMecanico(String nombreMecanico){
424         Boolean existe = false;
425         for (Mecanico mec: mecanicos){
426             if (mec.nombre.equals(nombreMecanico)){
427                 existe = true;
428                 break;
429             }
430         }
431         return existe;
432     }
433 
434 
435     /*
436      Métodos para gestionar objetos de la clase Oferta.
437      */
438 
439     // El método crearOferta crea un objeto oferta, le asigna los datos que devuelve el método nuevaOferta
440     // y guarda dicho objeto en el HashMap ofertas.
441 
442     public void crearOferta(){
443         Oferta oferta = new Oferta();
444         oferta = oferta.nuevaOferta();
445         ofertas.put(oferta.getDescripcion(), oferta);
446     }
447 
448 
449     // El método ofertaACliente pide los datos dni y descripción, con ello localiza los objetos cliente y oferta.
450     // Incluye la oferta al correspondiente ArrayList del cliente si el método noOfertada le ha devuelto
```

```
451     // que no lo estaba ya. Por último, le da fecha a la oferta cuando es presentada si el
    método haPasadounAnno
452     // le devuelve que ha sido presentada hace más de un año.
453
454     public void ofertaACliente(){
455         System.out.println("Dni cliente: ");
456         String dni = Menu.sc.nextLine();
457         System.out.println("Descripcion oferta: ");
458         String desc = Menu.sc.nextLine();
459         Cliente cl = clientes.get(dni);
460         Oferta ofe = ofertas.get(desc);
461         if (cl != null && ofe !=null){
462             Oferta yaOfertada = noOfertada(cl, ofe);
463             if (yaOfertada == null){
464                 if (cl.getOfertas()==null){
465                     cl.setOfertas(new ArrayList<Oferta>());
466                 }
467                 cl.getOfertas().add(ofe);
468             } else{
469                 if (haPasadounAnno(yaOfertada)){
470                     yaOfertada.setFechaPresentada(Calendar.getInstance());
471                 } else{
472                     System.out.println("Oferta ya presentada hace menos de un anno");
473                 }
474             }
475         } else{
476             System.out.println("No existe cliente u oferta");
477         }
478     }
479
480
481     // El método noOfertada comprueba, mediante la variable descripcion, si un objeto
    oferta
482     // se encuentra en el ArrayList ofertas de un cliente.
483
484     private Oferta noOfertada(Cliente cl, Oferta ofe){
485         if (cl.getOfertas()==null){
486             return null;
487         }
488         for (Oferta ofeCl: cl.getOfertas()){
489             if (ofeCl.getDescripcion().equals(ofe.getDescripcion())){
490                  return ofeCl;
491                 }
492         }
493         return null;
494     }
495
496
497     // El método haPasadounAnno es de tipo boolean, comprueba con el atributo
    FechaPresentada
498     // si una oferta ha sido presentada hace menos de un año, y devuelve true o false.
499
500     private Boolean haPasadounAnno(Oferta yaOfertada){
501         Calendar haceUnAnno = Calendar.getInstance();
502         haceUnAnno.add(Calendar.YEAR, -1);
503         if (yaOfertada.getFechaPresentada() == null ||
    yaOfertada.getFechaPresentada().before(haceUnAnno)){
504             return true;
505         }
506         return false;
507     }
508
```

```
509
510     // El método enviarOfertas recorre el HashMap clientes y a cada elemento le aplica el
    método enviarOfertas
511     // de la clase Cliente, que imprime los datos de las ofertas de cada cliente
512     // incluidas en su ArrayList ofertas.
513
514     public void enviarOfertas(){
515         for (Cliente cl: clientes.values()){
516             System.out.println(cl.enviarOfertas());
517         }
518     }
519
520
521     // El método pasarItv solicita una matricula para localizar un objeto vehiculo,
    comprueba
522     // si está pasando la ITV, y si no es así, le da valor true a la variable de tipo
    boolean aPasarItv.
523     // Si ya está pasando la ITV, muestra el mensaje de que la está pasando.
524
525     public void pasarItv(){
526         String matricula;
527         System.out.print("Matricula a pasar Itv: ");
528         matricula = Menu.sc.nextLine();
529         Vehiculo vehiculo = buscarVehiculo(matricula);
530         if (vehiculo!=null){
531             if (vehiculo.getItv()!=null && vehiculo.getItv().isaPasarItv()){
532                 System.out.println("Vehiculo ya esta pasando la itv");
533             } else{
534                 vehiculo.getItv().setaPasarItv(true);
535             }
536         } else{
537             System.out.println("Vehiculo no existe");
538         }
539     }
540
541
542     // El método modificarEstadoItv solicita una matricula para localizar un objeto
    vehiculo, comprueba
543     // si está pasando la ITV, y si es así, solicita indicar los trabajos realizados o
    pendientes de realizar
544     // para incluirlos en el ArrayList de clase Trabajo.
545
546     public void modificarEstadoItv(){
547         String matricula;
548         System.out.print("Matricula a modificar estado Itv: ");
549         matricula = Menu.sc.nextLine();
550         Vehiculo vehiculo = buscarVehiculo(matricula);
551         if (vehiculo!=null){
552             Itv vhItv = vehiculo.getItv();
553             if (vhItv==null || !vehiculo.getItv().isaPasarItv()){
554                 System.out.println("Vehiculo no esta pasando la itv. Dar de alta en Itv");
555             } else{
556                 System.out.print(Enumerados.menuMotivosVisita() + "\n");
557                 System.out.print("Introduzca numero de trabajos marcados en la Itv
    separados por blanco: ");
558                 String trbMenu = Menu.sc.nextLine();
559                 ArrayList<Trabajo> trbLista = Enumerados.setArrayTrabajos(trbMenu);
560                 vhItv.setaReparar(trbLista);
561                 System.out.print("Indique si ya estan reparados. 0.No reparado
    1.Reparado");
562                 Integer reparado = Menu.sc.nextInt();
563                 if (reparado>0){
```

```
564                     vhItv.setReparado(true);
565                 } else{
566                     vhItv.setReparado(false);
567                 }
568             }
569         } else{
570             System.out.println("Vehiculo no existe");
571         }
572     }
573 }
```