```java
/**
 * @ (#) GestionFicha.java
 *
 * Clase GestionFicha.
 * En GestionFicha se implementan todos los métodos relacionados con Ficha y FichaReparacion.
 * Esta clase tiene dos HashMap fichas y fichasRE donde incluimos todos los objetos que necesitemos guardar
 * de Ficha y FichaReparacion.
 *
 * @author Yassine Marroun
 * @version 1.00 2018/05/04
 */
package gestion;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.HashMap;
import java.util.Map;
import modelo.bd.Cliente;
import modelo.bd.Electrodomestico;
import modelo.bd.Ficha;
import modelo.bd.FichaReparacion;
import modelo.bd.Usuario;
import utilidades.Enumerados;
import utilidades.Utiles;

public class GestionFicha {

    private Map<Integer, Ficha> fichas = new HashMap<Integer, Ficha> ();
    private Map<Integer, FichaReparacion> fichasRE = new HashMap<Integer, FichaReparacion> ();

    // Métodos para gestionar las FICHAS de compra.

    /*
    El método numeroNuevaFicha comprueba el número de elementos que tiene el mapa fichas,
    y lo devuelve sumándole uno para que le asignemos ese número a la nueva ficha que creemos.
     */
    public int numeroNuevaFicha() {
        if (fichas!=null && fichas.size()>0){
           return fichas.size() + 1;
        }else{
           return 1;
        }
    }


    /*
    El método darFecha únicamente devuelve la fecha actual mediante la clase Calendar.
     */
    public Calendar darFecha(){
        Calendar fechaFic = Calendar.getInstance();
        fechaFic.getTime();
        return fechaFic;
    }


    /*
    El método compraCliente solicita un dni, localiza en el mapa al cliente que le corresponde ese campo
    y llama al método electrosAComprar pasándole dicho cliente.
```

```
58      */
59     public void compraCliente(){
60         System.out.println("Introduzca el DNI del cliente para realizar la compra:  ");
61         String dni = Utiles.sc.nextLine();
62         Cliente cl = GestionTienda.clientes.get(dni);
63         electrosAComprar(cl);
64     }
65 
66 
67     /*
68     El método electrosAComprar crea una ficha de compra, le asigna un número y la fecha actual,
69     solicita el modelo que se desea comprar y la cantidad de dicho artículo.
70     Mediante el método clone, crea una nueva instancia del electrodomestico que ha localizado
71     con ese modelo en el mapa electrodomesticos.
72     Llama al método cuentaStock pasándole ambos datos, cantidad y electrodomestico. Si la comprobación en cuanto
73     al número de existencias es correcta, se incluye una instancia de ese electrodomestico a la lista
74     correspondiente del cliente.
75     Le pasa datos al método rellenarFicha para terminar de incluir los datos de compra en la ficha.
76     Se guarda en una variable total el precio final de sumar todos los artículos comprados.
77     El método permite incluir más electrodomésticos si es necesario, únicamente se realiza una comprobación
78     respecto al modelo en la condición del while.
79     Por último, llama a los métodos asignarFichaACliente y tramitarFinanciacion pasándole cliente y ficha al primero,
80     y ficha al segundo. Por último, imprime la ficha.
81      */
82     private void electrosAComprar(Cliente cl) {
83 
84         try {
85         Integer existencias = 0;
86         Integer total = 0;
87         Ficha ficha = new Ficha();
88         ficha.setNumFicha(numeroNuevaFicha());
89         ficha.setFechaFic(darFecha());
90         System.out.println("Introduzca el MODELO comprado: ");
91         String modelo = Utiles.sc.nextLine();
92         while (!modelo.equals("S")) {
93             System.out.println("Introduzca el numero de articulos comprado: ");
94             String cant = Utiles.sc.nextLine();
95             Integer cantidad = Integer.parseInt(cant);
96             Electrodomestico electro = new Electrodomestico();
97             Object copia = GestionTienda.electrodomesticos.get(modelo).clone();
98             electro = (Electrodomestico) copia;
99             electro.setCantidad(cantidad);
100            if (cl != null && electro !=null){
101                cantidad = cantidad * -1;
102                existencias = cuentaStock(GestionTienda.electrodomesticos.get(modelo), cantidad);
103                if (cl.getElectrodomesticos()==null){
104                    cl.setElectrodomesticos(new ArrayList<Electrodomestico>());
105                }
106                if (existencias >= 0) {
107                    cl.getElectrodomesticos().add(electro);
108                    ficha = rellenarFicha(ficha, cl.getDni(), electro);
109                    total = total + electro.getCantidad() * electro.getPrecio();
110                }
111            } else{
```

```
112                 System.out.println("No existe Cliente o Electrodomestico.");
113             }
114             System.out.println("Introduzca el siguiente MODELO a comprar o S para terminar:
    ");
115             modelo = Utiles.sc.nextLine();
116         }
117 
118         asignarFichaACliente(cl, ficha);
119         tramitarFinanciacion(ficha);
120         System.out.println(ficha.toString());
121         System.out.println(" Total: " + total + " euros.");
122 
123         } catch(Exception e) {
124                 System.out.println("\nError en los datos introducidos. Intentelo de
    nuevo.");
125         }
126     }
127 
128 
129     /*
130     El método cuentaStock recibe un objeto de tipo Electrodomestico y el campo cantidad.
131     Actualiza el campo stock de dicho electrodomestico y también devuelve dicho dato para
    que se tenga
132     el número de existencias.
133      */
134     private Integer cuentaStock(Electrodomestico electro, Integer cantidad) {
135 
136         Integer cambio = electro.getStock();
137         cambio = cambio + cantidad;
138         if (cambio >= 0) {
139             electro.setStock(cambio);
140         } else {
141             System.out.println("Tenemos en stock " + electro.getStock() + " unidades del
    modelo " + electro.getModelo());
142         }
143         return cambio;
144     }
145 
146 
147     /*
148     El método tramitarFinanciacion recibe un objeto de tipo Ficha.
149     Pide indicar si se solicita financiación. En caso afirmativo, le asigna true al campo
    de tipo boolean financiar,
150     en caso de no necesitar financiación le da false. Por último, imprime la ficha.
151      */
152     public void tramitarFinanciacion(Ficha ficha) {
153 
154         System.out.println("\nSolicita FINANCIACION: ");
155         String financiar = Utiles.sc.nextLine();
156         if (financiar.equals("Si")) {
157             ficha.setFinanciar(true);
158         } else {
159             ficha.setFinanciar(false);
160         }
161         System.out.println(ficha.toString());
162     }
163 
164 
165     /*
166     El método confirmarFinanciacion pide que se introduzca un número de ficha, una vez
    localizada la ficha
167     mediante el método buscarFicha, asignamos a una variable de tipo Integer total el
```

```
    resultado que nos devuelve
168     el método calcularTotal habiéndole pasado la ficha. A continuación, solicita que se
    introduzca la nomina.
169     Realiza cálculos con la nomina para comprobar si es posible financiar la compra, en
    caso afirmativo
170     mantiene el campo financiar con true y guarda también el campo nomina, de lo contrario
    asigna a financiar false.
171      */
172     public void confirmarFinanciacion() {
173
174         System.out.print("Introduzca el numero de Ficha: ");
175         String numF = Utiles.sc.nextLine();
176         Integer numFicha = Integer.parseInt(numF);
177         Ficha ficha = buscarFicha(numFicha);
178         Integer total = calcularTotal(ficha);
179         if (ficha!=null){
180             System.out.println("Introduzca la NOMINA: ");
181             String nom = Utiles.sc.nextLine();
182             Integer nomina = Integer.parseInt(nom);
183             if (total/60 <= (nomina/100)*15) {
184                 ficha.setFinanciar(true);
185                 ficha.setNomina(nomina);
186             } else {
187                 ficha.setFinanciar(false);
188             }
189             System.out.println(ficha.toString());
190             System.out.println(" Total: " + total + " euros.");
191         } else {
192             System.out.println("Ficha no existente.");
193         }
194     }
195
196
197     /*
198     El método calcularTotal recibe una ficha. Mediante un bucle for each, recorre la lista
    de electrodomesticos
199     que contiene la ficha. Calcula el precio final de la compra sumando el precio de cada
    electrodomestico
200     multiplicado por la cantidad que se haya adquirido.
201     Por último, devuelve el resultado obtenido en la variable total.
202      */
203     private Integer calcularTotal(Ficha ficha) {
204
205         Integer total = 0;
206         for (Electrodomestico el: ficha.getElectrodomesticos()) {
207             total = total + el.getCantidad() * el.getPrecio();
208         }
209         return total;
210     }
211
212
213     /*
214     El método rellenarFicha recibe un objeto Ficha, un campo dniCliente y un objeto
    Electrodomestico.
215     Incluye dicho electrodomestico a la lista de electrodomésticos de la ficha
216     y por ultimo guarda la ficha en el mapa fichas.
217      */
218     public Ficha rellenarFicha(Ficha ficha, String dniCliente, Electrodomestico electro){
219
220         ficha.setDniCliente(dniCliente);
221         if (ficha.getElectrodomesticos()==null){
222             ficha.setElectrodomesticos(new ArrayList<Electrodomestico>());
```

```
223         }
224         ficha.getElectrodomesticos().add(electro);
225         fichas.put(ficha.getNumFicha(), ficha);
226         return ficha;
227     }
228 
229 
230     /*
231     El método asignarFichaACliente va a recibir un objeto de tipo Cliente y otro de tipo Ficha.
232     Únicamente va a incluir dicha ficha en la lista de fichas correspondiente al cliente.
233      */
234     public void asignarFichaACliente(Cliente clienteFicha, Ficha ficha) {
235 
236         if (clienteFicha.getFichas()==null){
237             clienteFicha.setFichas(new ArrayList<Ficha>());
238         }
239         clienteFicha.getFichas().add(ficha);
240     }
241 
242 
243     /*
244     El método buscarFicha recibe una variable numFicha con la que localiza
245     y devuelve la ficha que corresponda del mapa fichas.
246      */
247     public Ficha buscarFicha(Integer numFicha){
248         return fichas.get(numFicha);
249     }
250 
251 
252     /*
253     El método buscarFichaPantalla solicita el dato para numFicha.
254     Llama al método buscarFicha pasándole dicha variable, e imprime en pantalla los datos de la ficha
255     que le devuelva dicho método.
256      */
257     public void buscarFichaPantalla(){
258         System.out.print("Buscar Ficha por su numero: ");
259         String numF = Utiles.sc.nextLine();
260         Integer numFicha = Integer.parseInt(numF);
261         Ficha ficha = buscarFicha(numFicha);
262         if (ficha!=null){
263              System.out.println(ficha.toString());
264         } else {
265             System.out.println("Ficha no existente.");
266         }
267     }
268 
269 
270     /*
271     El método buscarFichaPorDNI solicita un dni, recorre el mapa fichas e imprime aquella ficha
272     que contenga el mismo campo dni.
273      */
274     public void buscarFichaPorDNI(){
275         String dni;
276         boolean existe = false;
277         System.out.print("Buscar Ficha por el DNI del Cliente: ");
278         dni = Utiles.sc.nextLine();
279         for(Ficha ficha: fichas.values()) {
280             if (ficha.getDniCliente().equals(dni)){
281                 existe = true;
```

```
282                 System.out.println(ficha.toString());
283             }
284         }
285         if (!existe){
286             System.out.println("No existen Fichas de dicho Cliente.");
287         }
288     }
289 
290 
291     /*
292     El método visualizarFichas imprime en pantalla todos los objetos ficha que tenemos en
    el mapa fichas.
293      */
294     public void visualizarFichas() {
295         System.out.println("Listado de fichas.");
296 
297         for(Ficha fic: fichas.values()) {
298             System.out.println(fic.toString());
299         }
300     }
301 
302 
303     // Métodos para gestionar las FICHAS de REPARACION.
304 
305     /*
306     El método numeroNuevaFichaRE comprueba el número de elementos que tiene el mapa
    fichasRE,
307     y lo devuelve sumándole uno para que le asignemos ese número a la nueva fichaRE que
    creemos.
308      */
309     public int numeroNuevaFichaRE() {
310         if (fichasRE!=null && fichasRE.size()>0){
311            return fichasRE.size() + 1;
312         }else{
313            return 1;
314         }
315     }
316 
317 
318     /*
319     El método repararCliente solicita que se introduzca un dni con el que se localiza a un
    cliente
320     en el mapa clientes. Pide un número de ficha de compra.
321     Llama al método comprobarFechaFic pasándole el objeto Cliente y la variable numFicha.
322     Y llama también a electrosAReparar pasándole el cliente.
323      */
324     public void repararCliente(){
325         System.out.println("Introduzca el DNI del Cliente: ");
326         String dni = Utiles.sc.nextLine();
327         Cliente cl = GestionTienda.clientes.get(dni);
328         System.out.println("Introduzca el NUMERO de Ficha: ");
329         String numF = Utiles.sc.nextLine();
330         Integer numFicha = Integer.parseInt(numF);
331         comprobarFechaFic(cl, numFicha);
332         electrosAReparar(cl);
333     }
334 
335 
336     /*
337     El método comprobarFechaFic recibe un cliente y una variable numFicha.
338     Recorre el mapa fichas en busca del elemento en el que coincida primero el campo
    dniCliente
```

```
339     y a continuación numFicha.
340     Una vez localizada la ficha, comprueba si su fecha tiene menos de dos años.
341     En casa de ser una ficha que tiene menos de dos años imprime un mensaje de reparación
    gratuita,
342     en caso contrario, la reparación tendrá un coste.
343      */
344     private void comprobarFechaFic(Cliente cl, Integer numFicha) {
345
346         for(Ficha ficha: fichas.values()) {
347             if (ficha.getDniCliente().equals(cl.getDni())) {
348                 if (ficha.getNumFicha().equals(numFicha)) {
349                     Calendar dosAnnos = Calendar.getInstance();
350                     dosAnnos.add(Calendar.YEAR, -2);
351                     if(ficha.getFechaFic().before(dosAnnos)) {
352                         System.out.println("Compra realizada hace mas de dos años. Importe
    a pagar en funcion de la reparacion.");
353                     } else {
354                         System.out.println("\nReparacion GRATUITA.\n");
355                     }
356                 }
357             } else {
358                 System.out.println("\nDatos incorrectos. No es posible crear Ficha de
    Reparacion.\n");
359                 break;
360             }
361         }
362     }
363
364
365     /*
366     El método electrosAReparar crea una instancia de FichaReparacion, le asigna un número y
    la fecha actual,
367     solicita el modelo de electrodoméstico que se desea reparar. Incluye la instancia
    creada de ese Electrodomestico
368     en la lista de electrodomesticos que tiene la fichaRE.
369     El método permite incluir más electrodomésticos si es necesario, únicamente se realiza
    una comprobación
370     respecto al modelo en la condición del while.
371     Para terminar de completar los datos de la ficha de reparación llama al método
    rellenarFichaRE
372     pasándole como parámetro fichaRE y el dni del cliente que electrosAReparar ha recibido.
373     Por último, imprime la fichaRE que devuelve rellenarFichaRE.
374      */
375     private void electrosAReparar(Cliente cl) {
376
377         try {
378         FichaReparacion fichaRE = new FichaReparacion();
379         fichaRE.setNumFicha(numeroNuevaFichaRE());
380         fichaRE.setFechaFic(darFecha());
381         System.out.println("Introduzca el MODELO a REPARAR: ");
382         String modelo = Utiles.sc.nextLine();
383         while (!modelo.equals("S")) {
384             Electrodomestico electro = GestionTienda.electrodomesticos.get(modelo);
385             if (cl != null && electro !=null){
386                 if (fichaRE.getElectrodomesticos()==null){
387                     fichaRE.setElectrodomesticos(new ArrayList<Electrodomestico>());
388                 }
389                 fichaRE.getElectrodomesticos().add(electro);
390             } else{
391                 System.out.println("No existe Cliente o Electrodomestico.");
392             }
393             System.out.println("Introduzca el siguiente MODELO a REPARAR o S para terminar:
```

```
");
394             modelo = Utiles.sc.nextLine();
395         }
396         fichaRE = rellenarFichaRE(fichaRE, cl.getDni());
397         System.out.println(fichaRE.toStringRE());
398 
399         } catch(Exception e) {
400                 System.out.println("\nError en los datos introducidos. Intentelo de
nuevo.");
401         }
402     }
403 
404 
405     /*
406     El método rellenarFichaRE recibe un objeto FichaReparacion y un campo dniCliente.
407     Solicita datos para los campos nombreTecnico, piezas y observaciones para asignarlos a
la fichaRE recibida.
408     El campo estado, de tipo enumerado, al crear la ficha se establece como PENDIENTE.
409     Llama al método asignarTecnicoFicha pasándole fichaRE y el campo nombreTecnico.
410     Por último, guarda la ficha de reparación en el mapa fichasRE.
411      */
412     public FichaReparacion rellenarFichaRE(FichaReparacion fichaRE, String dniCliente){
413 
414         fichaRE.setDniCliente(dniCliente);
415         System.out.println("Introduzca el nombre del TECNICO: ");
416         String nombreTecnico = Utiles.sc.nextLine();
417         asignarTecnicoAFicha(fichaRE, nombreTecnico);
418         System.out.println("Introduzca las PIEZAS necesarias para esta Ficha de Reparacion:
");
419         String piezas = Utiles.sc.nextLine();
420         fichaRE.setPiezas(piezas);
421         System.out.println("Introduzca las OBSERVACIONES para esta Ficha de Reparacion: ");
422         String observaciones = Utiles.sc.nextLine();
423         fichaRE.setObservaciones(observaciones);
424         fichaRE.setEstado(Enumerados.EstadoReparacion.PENDIENTE);
425         fichasRE.put(fichaRE.getNumFicha(), fichaRE);
426         return fichaRE;
427     }
428 
429 
430     /*
431     El método asignarTecnicoAFicha recibe una instancia de FichaReparacion y una variable
nombreTecnico,
432     mediante el método buscarTecnico comprueba si existe dicho técnico, en caso afimativo,
433     asigna dicho campo nombreTecnico en la fichaRE, de lo contrario imprime un mensaje
donde se hace saber
434     que no se encuentra el nombre del técnico.
435      */
436     public void asignarTecnicoAFicha(FichaReparacion fichaRE, String nombreTecnico){
437         Boolean existe = false;
438         if (fichaRE!=null){
439             existe = buscarTecnico(nombreTecnico);
440             if (existe){
441                 fichaRE.setNombreTecnico(nombreTecnico);
442             } else{
443                 System.out.println("No existe el tecnico: " + nombreTecnico);
444             }
445         } else {
446             System.out.println("No existe la ficha.");
447         }
448     }
449 
```

```
450
451     /*
452     El método buscarTecnico recibe una variable nombreTecnico. Recorre la lista tecnicos,
453     en caso de encontrar el objeto en el que coincide el campo nombre, devuelve true,
454     de lo contrario devuelve false.
455      */
456     private Boolean buscarTecnico(String nombreTecnico){
457         Boolean existe = false;
458         for (Usuario tec: GestionTienda.tecnicos){
459             if (tec.getNombre().equals(nombreTecnico)){
460                 existe = true;
461                 break;
462             }
463         }
464         return existe;
465     }
466
467
468     /*
469     El método buscarFichaRE recibe una variable numFichaRE con la que localiza y devuelve
    la fichaRE
470     que corresponda del mapa fichasRE.
471      */
472     public FichaReparacion buscarFichaRE(Integer numFichaRE){
473         return fichasRE.get(numFichaRE);
474     }
475
476
477     /*
478     El método darEstadoConsola muestra los posibles valores de tipo enumerado que se pueden
    dar al campo estado.
479     Y devuelve aquel que se selecciona por teclado.
480      */
481     public Enumerados.EstadoReparacion darEstadoConsola(){
482         System.out.print(Enumerados.menuEstadoReparacion() + "\n");
483         String iEs = Utiles.sc.nextLine();
484         Integer iEstado = Integer.parseInt(iEs);
485         return Enumerados.EstadoReparacion.values()[iEstado];
486     }
487
488
489     /*
490     El método motivoParado muestra los posibles valores de tipo enumerado que se pueden dar
    al campo parado.
491     Devolviendo aquel que se selecciona por teclado.
492      */
493     public Enumerados.Parado motivoParado(){
494         System.out.print(Enumerados.menuParado() + "\n");
495         String iPa = Utiles.sc.nextLine();
496         Integer iParado = Integer.parseInt(iPa);
497         return Enumerados.Parado.values()[iParado];
498     }
499
500
501     /*
502     El método cambiarEstado solicita que se introduzca por teclado el campo numFichaRE,
503     localiza la ficha de reparación mediante el método buscarFichaRE. Asigna el nuevo
    estado
504     mediante la devolución que obtiene del método darEstadoConsola. Si el nuevo estado es
    PARADO,
505     llama al método motivoParado para dar el valor correspondiente a la variable parado.
506     Por último, modifica también el campo observaciones con aquello que se introduzca por
```

```
   teclado.
507      */
508     public void cambiarEstado(){
509         System.out.print("Introduzca el numero de Ficha a cambiar: ");
510         String numF = Utiles.sc.nextLine();
511         Integer numFichaRE = Integer.parseInt(numF);
512         FichaReparacion fichaRE = buscarFichaRE(numFichaRE);
513         if (fichaRE!=null){
514             System.out.print("Introduzca el nuevo ESTADO de la ficha: ");
515             Enumerados.EstadoReparacion estado = darEstadoConsola();
516             fichaRE.setEstado(estado);
517             if (estado == Enumerados.EstadoReparacion.PARADO) {
518                 System.out.print("Introduzca el motivo por el que esta PARADA la ficha: ");
519                 Enumerados.Parado iParado = motivoParado();
520                 fichaRE.setParado(iParado);
521             }
522             System.out.print("Editar las OBSERVACIONES de la ficha: ");
523             String obs = Utiles.sc.nextLine();
524             fichaRE.setObservaciones(obs);
525             System.out.println(fichaRE.toStringRE());
526         } else {
527             System.out.println("Ficha no existente.");
528         }
529     }
530
531
532     /*
533     El método visualizarFichasRE imprime en pantalla todos los elementos fichaRE que
   tenemos en el mapa fichasRE.
534      */
535     public void visualizarFichasRE() {
536         System.out.println("Listado de Fichas de REPARACION.");
537
538         for(FichaReparacion fic: fichasRE.values()) {
539             System.out.println(fic.toStringRE());
540         }
541     }
542
543
544     /*
545     El método visualizaFichasRETecnico solicita que se introduzca un nombre para la
   variable nomTecnico,
546     recorre el mapa fichasRE e imprime aquellas fichas de reparación en las que coincida el
   campo nombreTecnico
547     y cuyo estado sea distinto de TERMINADO.
548      */
549     public void visualizaFichasRETecnico() {
550         boolean existe = false;
551         System.out.println("Introduzca el nombre del TECNICO: ");
552         String nomTecnico = Utiles.sc.nextLine();
553         System.out.println("Listado de fichas pendientes del Tecnico: " + nomTecnico);
554         for(FichaReparacion fic: fichasRE.values()) {
555             if (fic.getNombreTecnico().equals(nomTecnico) && fic.getEstado() !=
   Enumerados.EstadoReparacion.TERMINADO){
556                 System.out.println(fic.toStringRE());
557                 existe = true;
558             }
559         }
560         if (!existe){
561             System.out.println("No existen fichas de este Tecnico.");
562         }
563     }
```

```
564
565
566     /*
567     El método historialTecnico solicita que se introduzca un nombre para la variable
    nomTecnico,
568     recorre el mapa fichasRE e imprime aquellas fichas de reparación en las que coincida el
    campo nombreTecnico
569     y cuyo estado sea TERMINADO.
570      */
571     public void historialTecnico() {
572         boolean existe = false;
573         System.out.println("Introduzca el nombre del TECNICO: ");
574         String nomTecnico = Utiles.sc.nextLine();
575         System.out.println("Historial de fichas del Tecnico: " + nomTecnico);
576         for(FichaReparacion fic: fichasRE.values()) {
577             if (fic.getNombreTecnico().equals(nomTecnico) && fic.getEstado() ==
    Enumerados.EstadoReparacion.TERMINADO){
578                 System.out.println(fic.toStringRE());
579                 existe = true;
580             }
581         }
582         if (!existe){
583             System.out.println("No existen fichas de este Tecnico.");
584         }
585     }
586
587
588     /*
589     El método listarPiezas recorre el mapa fichasRE e imprime el número de ficha y el campo
    piezas
590     de aquellas fichas de reparación cuyo estado sea PENDIENTE.
591      */
592     public void listarPiezas() {
593         boolean existe = false;
594         System.out.println("Listado de PIEZAS a pedir: ");
595         for(FichaReparacion fic: fichasRE.values()) {
596             if (fic.getEstado() == Enumerados.EstadoReparacion.PENDIENTE){
597                 System.out.println("Numero de Ficha: " + fic.getNumFicha() + "  Piezas: " +
    fic.getPiezas());
598                 existe = true;
599             }
600         }
601         if (!existe){
602             System.out.println("No existen Fichas PENDIENTES para solicitar Piezas.");
603         }
604     }
605
606
607     /*
608     El método solicitarConfirmacion recorre el mapa fichasRE y muestra en pantalla aquellas
    fichas de reparación
609     cuyo estado sea PARADO y el motivo de ello sea CONFIRMAR_CLIENTE.
610      */
611     public void solicitarConfirmacion() {
612         boolean existe = false;
613         System.out.println("Solicitar la Confirmacion a los Clientes de las siguientes
    Fichas de Reparacion: ");
614         for(FichaReparacion fic: fichasRE.values()) {
615             if (fic.getParado() == Enumerados.Parado.CONFIRMAR_CLIENTE){
616                 System.out.println(fic.toStringRE());
617                 existe = true;
618             }
```

```
619         }
620         if (!existe){
621             System.out.println("No existen Fichas a Confirmar por Cliente.");
622         }
623     }
624 
625 
626     /*
627     El método fichasEnProceso recorre el mapa fichasRE e imprime en pantalla aquellas fichas de reparación
628     que estén en estado EN_PROCESO.
629      */
630     public void fichasEnProceso() {
631         boolean existe = false;
632         System.out.println("Listado de Fichas que estan En Proceso: ");
633         for(FichaReparacion fic: fichasRE.values()) {
634             if (fic.getEstado() == Enumerados.EstadoReparacion.EN_PROCESO){
635                 System.out.println(fic.toStringRE());
636                 existe = true;
637             }
638         }
639         if (!existe){
640             System.out.println("No existen Fichas En Proceso.");
641         }
642     }
643 
644 
645     /*
646     El método historialElectro solicita que se introduzca un modelo de electrodomestico.
647     Recorre el mapa fichasRE en busca de aquellas fichas de reparación que contengan ese electrodoméstico
648     en su lista. Finalmente imprime el número de ficha y los campos piezas y observaciones de todas aquellas fichas
649     que tengan incluido el electrodoméstico que se busca.
650      */
651     public void historialElectro() {
652         boolean existe = false;
653         System.out.println("Introduzca el MODELO de electrodomestico: ");
654         String modelo = Utiles.sc.nextLine();
655         System.out.println("Historial del Electrodomestico: " + modelo);
656         for(FichaReparacion ficRE: fichasRE.values()) {
657             for (Electrodomestico el: ficRE.getElectrodomesticos()) {
658                 if (el.getModelo().equals(modelo)){
659                     System.out.println("Numero de Ficha: " + ficRE.getNumFicha() + " Piezas: " + ficRE.getPiezas() +
660                             " Observaciones: " + ficRE.getObservaciones());
661                     existe = true;
662                 }
663             }
664         }
665         if (!existe){
666             System.out.println("No existe historial de este Electrodomestico.");
667         }
668     }
669 
670 
671     // Métodos para gestionar la POSTVENTA.
672 
673     /*
674     El método devolucionCliente solicita que se dé un dni, un modelo y la cantidad a devolver de dicho modelo.
675     Localiza tanto al cliente como al electrodomestico en los correspondientes mapas
```

```
   clientes y electrodomesticos
676     gracias a las variables dni y modelo introducidas.
677     A continuación, recorre el mapa fichas en busca de aquellas en las que coincida el dni
   dado
678     con el campo dniCliente de la ficha. Una vez tiene la ficha, comprueba que no haya
   pasado más de tres meses
679     desde que se hizo la compra, si es así, llama al método eliminarElectro pasándole la
   lista de electrodomesticos
680     de la ficha localizada y queremos modificar, el modelo de electrodoméstico y la
   cantidad a devolver
681     de dicho modelo. También llama al método cuentaStock pasándole el electrodomestico y la
   variable cantidad.
682     El método permite incluir más electrodomésticos en la devolución si es necesario,
   únicamente se realiza una comprobación respecto al modelo en la condición del while.
683     Por último, imprime como quedan los datos de la ficha tras la devolución.
684      */
685     public void devolucionCliente(){
686
687         try {
688         Ficha fichaEncontrada = new Ficha();
689         System.out.println("Introduzca el DNI del cliente para realizar la devolucion: ");
690         String dni = Utiles.sc.nextLine();
691         System.out.println("Introduzca el MODELO a devolver: ");
692         String modelo = Utiles.sc.nextLine();
693         while (!modelo.equals("S")) {
694             System.out.println("Introduzca el numero de articulos a devolver: ");
695             String cant = Utiles.sc.nextLine();
696             Integer cantidad = Integer.parseInt(cant);
697             Cliente cl = GestionTienda.clientes.get(dni);
698             Electrodomestico electro = GestionTienda.electrodomesticos.get(modelo);
699                 if (cl != null){
700                     for(Ficha ficha: fichas.values()) {
701                         if (ficha.getDniCliente().equals(dni)){
702                             fichaEncontrada = ficha;
703                             Calendar tresMeses = Calendar.getInstance();
704                             tresMeses.add(Calendar.MONTH, +3);
705                                 if(ficha.getFechaFic().before(tresMeses)) {
706                                     eliminarElectro(ficha.getElectrodomesticos(), modelo,
   cantidad);
707                                     cuentaStock(electro, cantidad);
708                                 } else {
709                                     System.out.println("Esta compra se ha realizado hace
   mas de TRES meses.");
710                                 }
711                         }
712                     }
713                 } else{
714                     System.out.println("No existe Cliente o Electrodomestico.");
715                 }
716             System.out.println("Introduzca el siguiente MODELO a devolver o S para
   terminar: ");
717             modelo = Utiles.sc.nextLine();
718         }
719         System.out.println("\nDevolucion Correcta.");
720         System.out.println(fichaEncontrada.toString());
721
722         } catch(Exception e) {
723                 System.out.println("\nError en los datos introducidos. Intentelo de
   nuevo.");
724         }
725     }
726
```

```
727
728     /*
729     El método eliminarElectro recibe un ArrayList que almacena objetos de tipo
    Electrodomestico,
730     una variable modelo y otra variable cantidad.
731     Recorre la lista en busca de la instancia electrodomestico en la que coincida el
    modelo.
732     Una vez localizado dicho objeto electrodomestico, actualiza su campo cantidad
    restándole la variable cantidad
733     que se le ha pasado como parámetro. En caso de que la cantidad quede igual a 0,
734     se elimina dicho electrodomestico de la lista.
735      */
736     private void eliminarElectro(ArrayList<Electrodomestico> electroLista, String modelo,
    Integer cantidad){
737
738         Electrodomestico electroEncontrado = null;
739         for(Electrodomestico el: electroLista) {
740             if(el.getModelo().equals(modelo)){
741                 electroEncontrado = el;
742                 break;
743             }
744         }
745         if(electroEncontrado != null) {
746             Integer nuevaCant = electroEncontrado.getCantidad();
747             nuevaCant = nuevaCant - cantidad;
748             if (nuevaCant >= 0) {
749                 electroEncontrado.setCantidad(nuevaCant);
750                 if(nuevaCant == 0) {
751                     electroLista.remove(electroEncontrado);
752                 }
753             } else {
754                 System.out.println("Se intenta devolver una cantidad mayor a la
    comprada.");
755             }
756         } else {
757             System.out.println("Modelo no encontrado.");
758         }
759     }
760 }
```