

Práctica de Programación Orientada a Objetos – Curso 2022 / 2023

Angela Alexandra GUZMÁN GARCÍA

Correo electrónico: angelagn50@gmail.com Telefono: 642154124

11 de mayo de 2023

1. Decisiones de diseño:

La aplicación se desarrolló en tres momentos, inicialmente se realizó presenta a través del planteamiento de la estructura de cada clase en lápiz y papel, así como el planteamiento del diagrama de casos de uso, de esta manera se generó una estructura inicial frente a la cual se busca dar respuesta a los planteamientos de la práctica. La aplicación está diseñada para que el programa siga ejecutándose y el usuario pueda navegar por el menú hasta que decida salir; en el método main el menú de opciones se implementó a través de un do-While el cual inicialmente da la bienvenida al usuario y propone 4 opciones de navegación, proveedor, cliente, administrados y salir de la aplicación, a medida que el usuario va eligiendo las opciones de menú y según el caso va avanzando por las diferentes tareas requeridas por la práctica en los tres puntos principales mostrando nuevamente el menú principal para evitar la terminación del programa hasta que el usuario así lo desee.

1.1. Como usar la aplicación

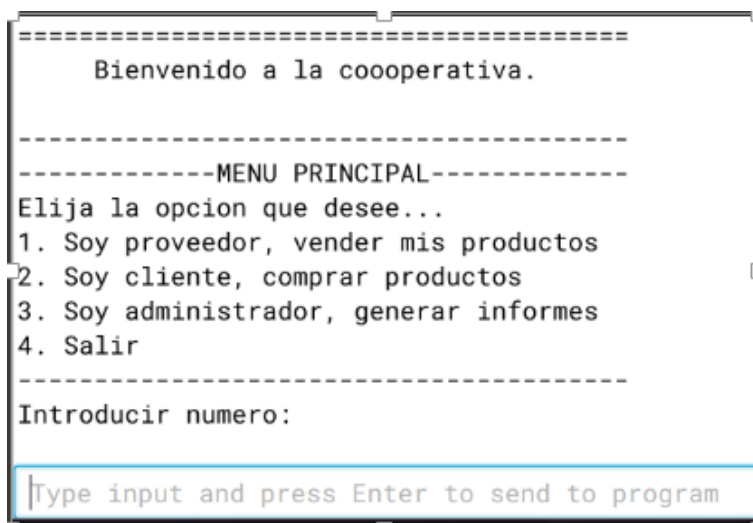


Figura 1: Menu bienvenido a la aplicación

El primer contacto de la aplicación con el usuario muestra un menú de bienvenida en el que se presentan 4 opciones según el usuario que se desee. Al introducir 1 se despliega la funcionalidad para el productor, la cual permite ingresar los productos que vayan a ser vendidos a la cooperativa.

```

Introducir numero:
1
-----
Bienvenido asociado
-----
-----
Ingrese el producto a añadir
-----
Ingrese el nombre del producto:
Manzana
Ingrese el precio del producto:
0.80
El valor ingresado no es valido, intente nuevamente:
0,80
Ingrese cantidad (toneladas) del producto:
2
¿Es perecedero? true/false:
true

```

Figura 2: Ingresar producto

En este ejemplo se agregan tres productos Manzana, Piña y Algodón.

```

-----
Ingrese el nombre del producto:
Algodon
Ingrese el precio del producto:
0,50
Ingrese cantidad (toneladas) del producto:
3
¿Es perecedero? true/false:
false
-----
¿Desea agregar mas productos?: si/no
-----
no
* Lista de productos agregados:

Manzana
Piña
Algodon
-----
Pagar al proveedor: 750.0 €.
=====
Bienvenido a la cooperativa.

-----
-----MENU PRINCIPAL-----
Elija la opcion que desee...
1. Soy proveedor, vender mis productos
2. Soy cliente, comprar productos
3. Soy administrador, generar informes
4. Salir
-----
Introducir numero:
Type input and press Enter to send to program

```

Figura 3: Lista de productos agregados y cantidad a pagar al proveedor

Como se observa en la imagen anterior, la aplicación ofrece una lista de productos agregados y el valor neto a pagar al proveedor por parte de la cooperativa por sus productos. Y para evitar que el programa termine, se presenta nuevamente el menú principal para que el usuario explore las siguientes funcionalidades.

```

-----
Introducir numero:
2
Lista de productos disponibles

* Lista de productos disponibles:

Producto: Manzana, precio: 3.4 €.
Producto: Pera, precio: 3.4 €.
Producto: Algodon, precio: 3.4 €.
Producto: Pimiento, precio: 3.4 €.
Producto: Arroz, precio: 3.4 €.
Producto: Cebolla, precio: 3.4 €.
Producto: Brocoli, precio: 3.4 €.
Indique el nombre del producto a comprar

Can only enter input while your program is run

```

Figura 4: Ingresar producto

La opción numero 2 del menú principal realiza las operaciones requeridas por los clientes a la hora de comprar productos, en este caso, se muestra una lista de productos disponibles, el usuario indica que producto o productos quiere comprar de la lista y se crea una lista llamada cesta en la que se guardan los productos que el cliente ha elegido como se muestra a continuación.

```

Indique el nombre del producto a comprar
Manzana
Desea agregar otro producto? si/no
si
Indique el nombre del producto a comprar
Algodon
Desea agregar otro producto? si/no
si
Indique el nombre del producto a comprar
Brocoli
Desea agregar otro producto? si/no
no
-----
Productos en la cesta.
-----
Manzana
Algodon
Brocoli
-----
Productos disponibles.
-----
Pera
Pimiento
Arroz
Cebolla
A que ciudad desea enviar sus productos?

Type input and press Enter to send to program

```

Figura 5: Productos en cesta y disponibles

De la misma manera, se presenta la lista de productos disponibles que el cliente no ha elegido. Seguido a esto, se solicita el nombre de la ciudad a enviar el producto para gestionar la logística correspondiente.

```

A que ciudad desea enviar sus productos?
Madrid
Ingrese distancia en kilometros para su envio.
160
Envio a Madrid a 160 Km, en 3 tramos de 50Km y 10Km en pequeña logística
El coste de la Gran logística es de: 1.125 Euros.

```

Figura 6: Cálculo de tramos y precio de logística

La aplicación también pide al usuario la distancia para el envío y calcula los tramos a recorrer en gran logística y la distancia requerida en pequeña logística. Luego, calcula el coste de la gran logística y pequeña logística según la distancia, a continuación se muestra un ejemplo de cálculo en caso de que la distancia sea menor a 50 km lo que corresponde a pequeña logística.

```

A que ciudad desea enviar sus productos?
Barcelona
Ingrese distancia en kilometros para su envio.
48
-----
Enviando producto a traves de pequeña logística.
-----
Distancia a recorrer: 48 Kilometros.
Coste Pequeña Logística: 144.0 Euros.
=====

```

Figura 7: Calculo para envio de productos

La opción 3 despliega un mensaje de bienvenida para el administrados y un menú de tres informes posibles a generar, de productos, de ventas y de rendimiento como se muestra a continuación.

```

-----
Introducir numero:
3
-----
Bienvenido administrador

-----
Elija la opcion que desee...
1. Generar informe de productos
2. Generar informe de ventas
3. Generar informe de rendimiento
-----
Introducir numero:
Type input and press Enter to send to program

```

Figura 8: Menu informes

Al elegir el informe de productos, la aplicación muestra un listado de productos y cantidad de ganancia por producto para el último año, como se muestra en la siguiente imagen.

```

-----
Introducir numero:
1
-----
INFORME DE PRODUCTOS

* Beneficios por producto:

Pimiento: 20 euros.
Pera: 34 euros.
Algodon: 95 euros.
Manzana: 56 euros.
Brocoli: 10 euros.
Cebolla: 7 euros.
Arroz: 27 euros.
=====

```

Figura 9: Informe de productos

Para ir al siguiente informe, hay que pasar nuevamente por el menú principal y elegir la opción de administrador nuevamente. La opción informe de ventas muestra la cantidad de toneladas vendidas por producto en el último año.

```

-----
Introducir numero:
2
-----
INFORME DE VENTAS

Cantidad de productos en toneladas vendidas el ultimo año:

Pimiento: 250 toneladas.
Pera: 20 toneladas.
Algodon: 120 toneladas.
Manzana: 220 toneladas.
Brocoli: 180 toneladas.
Cebolla: 280 toneladas.
Arroz: 267 toneladas.

```

Figura 10: Ingresar ventas

Para visualizar el informe de rendimiento, es necesario volver a ingresar la opción 3 en el menú principal y elegir el informe de rendimiento con la opción 3 nuevamente, la aplicación mostrará el informe de rendimiento el cual contiene el importe obtenido según productos, el importe obtenido según empresa de logística y el precio total de la logística en el último año.

```
3. General informe de rendimiento
-----
Introducir numero:
3
-----
INFORME DE RENDIMIENTO
Importe de productores

* Importe obtenido segun productor:

Jose: 2647 euros.
Luis: 2807 euros.
Rosa: 9435 euros.
Maria: 2034 euros.
Carmen: 1480 euros.
Antonio: 2340 euros.
Juan: 3456 euros.
Importe de empresas

* Importe obtenido segun empresa:

DHL: 4435 euros.
Transportes TM: 8034 euros.
Logis S.A: 4556 euros.
-----
El precio total de la pequeña logistica el ultimo
año fue: 6000 euros
```

Figura 11: Ingresar producto

2. Diagrama de clases

Las clases `PequenoProductor`, `GranProductor` y `ProductorFederado` heredan de la clase `Productor`. Y de la misma manera, las clases `PequenaLogistica` y `GranLogistica` heredan de una clase padre llamada `Logistica` la cual a su vez hereda de una clase llamada `Coste` y ésta, a su vez hereda de la clase `Producto`.

También se han construido clases que enlazan el funcionamiento de la aplicación como la clase Pedido, GestionPedidos y GestionProductos. Para el apartado 3 sobre informes, se han construido clases para cuatro tipos de informes las cuales son InformeEmpresa, InformeVentas, InformeProductor, InformeProductos.

La clase Cooperativa contiene el método main de la aplicación, para darle forma a la misma y para la interacción con el usuario se ha implementado una clase llamada imprimirMenu, la cual contiene las diferentes etapas de navegación del usuario por la aplicación.

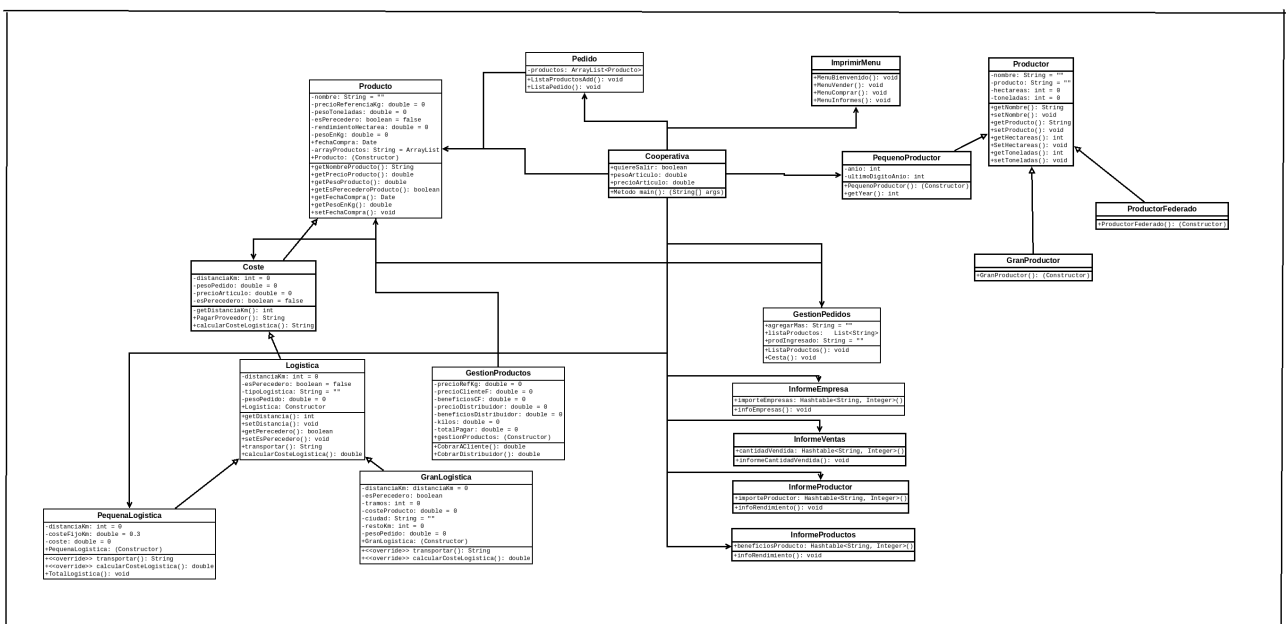


Figura 12: Diagrama de clases

3. Descripción de clases

```
1  import java.util.Scanner;
2  import java.util.ArrayList;
3
4
5  /**
6   * Clase Cooperativa que contiene el metodo main y las acciones que realiza el
7   * usuario al interactuar con la aplicacion.
8   * @author (Angela Alexandra Guzman Garcia)
9   * @version (001)
10  */
11  public class Cooperativa
12  {
13      public static void main(String[] args){
14          //Declara variables
15          boolean quiereSalir = false;
16          double pesoArticulo = 0.5;
17          double precioArticulo = 1.5; // Precio del articulo
18
19          do{
20              ImprimirMenu menu = new ImprimirMenu();
21              menu.MenuBienvenido();
22              Scanner sc = new Scanner(System.in);
23              System.out.println("Introducir numero: ");
24              int opcion = sc.nextInt();
25
26              switch(opcion){
27                  case 1:
28                      menu.MenuVender();
29
30                      Pedido pedido1 = new Pedido(new ArrayList<Producto>()); // crea
31                      objeto de tipo Pedido
32                      pedido1.ListaProductosAdd();
33                      System.out.println("* Lista de productos agregados:\n");
34                      pedido1.ListaPedido(); //Imprime la list
35                      Coste costePedido = new Coste(); //Objeto tipo coste, parametro
36                      kilometros
37
38                      String resultadoPago = costePedido.PagarProveedor(precioArticulo,
39                      pesoArticulo);
40                      System.out.println("-----");
41                      System.out.println(resultadoPago); //Imprime el total en euros a
42                      pagar al proveedor
43
44                      Coste pagar1 = new Coste();
45
46                      pagar1.PagarProveedor(precioArticulo, pesoArticulo);
47                      break;
48                  case 2:
49                      menu.MenuComprar();
50                      sc.nextLine();
51                      System.out.println("* Lista de productos disponibles:\n");
52                      GestionPedidos pedido2 = new GestionPedidos(); //crea objeto de
53                      tipo GestionPedidos
54                      pedido2.ListaProductos();
55                      pedido2.Cesta();
56
57                      //Envia los productos
58                      //Pide por consola la ciudad y distancia en kilometros
59                      System.out.println("A que ciudad desea enviar sus productos?");
60                      String ciudad = sc.nextLine();
61                      System.out.println("Ingrese distancia en kilometros para su envio."
62                      );
63              }
64          }
65      }
66  }
```

```

57         int km = sc.nextInt();
58         double costeProducto = 1.50;
59         PequenaLogistica pLogistica1 = new PequenaLogistica(km);
60         GranLogistica gLogistica1 = new GranLogistica(ciudad, km); //Crea
           objeto gran logistica
61         //Elige el tipo de logistica segun numero de kilometros
62         if(km <= 100){
63             System.out.println(pLogistica1.Transportar());
64             System.out.println(pLogistica1.calcularCosteLogistica());
65         }else{
66             System.out.println(gLogistica1.Transportar());
67             System.out.println(gLogistica1.calcularCosteLogistica(
                pesoArticulo, km, costeProducto));
68         }
69         break;
70     case 3:
71         // informes
72         menu.MenuInformes();
73         System.out.println("Introducir numero: ");
74         int opc = sc.nextInt();
75         switch(opc){
76             case 1:
77                 //INFORME DE PRODUCTOS
78                 System.out.println("
                    -----");
79                 System.out.println("INFORME DE PRODUCTOS \n      ");
80                 PequenoProductor pProductor1 = new PequenoProductor("Pedro"
                    , "Platano", 3, 1);
81                 pProductor1.MostrarPequeProductores();
82                 //crea objeto para mostrar el informe de productos
83                 InformeProductos informep = new InformeProductos();
84                 informep.infoRendimiento();
85                 break;
86             case 2:
87                 //INFORME DE VENTAS
88                 System.out.println("
                    -----");
89                 System.out.println("INFORME DE VENTAS \n      ");
90                 InformeVentas informes = new InformeVentas();
91                 informes.infoCantidadVendida();
92                 break;
93
94             case 3:
95                 //INFORME DE RENDIMIENTO3
96                 System.out.println("
                    -----");
97                 System.out.println("INFORME DE RENDIMIENTO ");
98                 //crea un objeto y llama a sus metodos
99                 InformeProductor informeR = new InformeProductor();
100                 System.out.println("Importe de productores\n");
101                 informeR.infoRendimiento();
102                 InformeEmpresa informeE = new InformeEmpresa();
103                 System.out.println("Importe de empresas\n");
104                 informeE.infoEmpresas();
105                 PequenaLogistica l1 = new PequenaLogistica(500);
106                 l1.TotalLogistica();
107                 break;
108         }
109         break;
110     case 4:
111         quiereSalir = true;
112         System.out.println("Hasta Pronto");
113         break;
114     default:
115         menu.MenuBienvenido();
116         break;

```



```

117         }
118     }while(quiereSalir != true);
119 }
120 }

```

El programa se inicia en la clase Cooperativa, que contiene el método principal main y se encarga de interactuar con el usuario. A través de menús, el usuario puede realizar diferentes acciones.

El programa utiliza varias clases para representar diferentes conceptos. La clase ImprimirMenu se encarga de imprimir los menús en la consola para que el usuario pueda seleccionar una opción. La clase Pedido representa un pedido y contiene una lista de productos. La clase Coste gestiona los costos y pagos relacionados con los pedidos y proveedores.

La clase GestionPedidos se encarga de gestionar los pedidos de productos disponibles en la cooperativa. Permite al usuario ver la lista de productos y agregarlos a una cesta. Además, se utilizan las clases PequeñaLogistica y GranLogistica para gestionar la logística de transporte, considerando distancias cortas y largas respectivamente.

También se incluyen clases como PequeñoProductor que representan a los pequeños productores, y clases de informes como InformeProductos, InformeVentas, InformeProductor y InformeEmpresa, que generan informes relacionados con productos, ventas, rendimiento y empresas respectivamente.

En general, el programa permite al usuario realizar acciones como vender productos, realizar pedidos, calcular costos de logística, generar informes y más. Cada clase desempeña un papel específico en el funcionamiento del sistema de gestión de la cooperativa, interactuando entre sí para proporcionar las funcionalidades requeridas.

```

1  import java.util.ArrayList;
2  import java.util.Date;
3
4  /**
5   * Clase Producto gestiona los productos
6   * en la cooperativa.
7   * @author (Angela Alexandra Guzman Garcia)
8   * @version (001)
9   */
10 public class Producto
11 {
12     // Declarar variables
13     private String nombre;
14     static double precioReferenciaKg;
15     private double pesoToneladas;
16     private boolean esPerecedero;
17     private double pesoEnKg;
18     public Date fechaCompra;
19
20     private ArrayList<String> arrayProductos;
21
22
23     /**
24      * Constructor for objects of class Producto
25      */
26     public Producto(String nombreArticulo, double precioArticulo,
27                     double pesoArticulo, boolean esPerecederoArticulo )
28     {
29         // initialise instance variables
30         this.nombre = nombreArticulo;
31         this.precioReferenciaKg = precioArticulo;
32         this.pesoToneladas = pesoArticulo;
33         this.esPerecedero = esPerecederoArticulo;
34     }
35     public Producto(double precioArticulo, double pesoArticulo)
36     {
37         // initialise instance variables
38         this.precioReferenciaKg = precioArticulo;
39         this.pesoToneladas = pesoArticulo;
40
41     }
42

```

```

43     /**Metodos get
44     */
45     public String getNombreProducto()
46     {
47         return this.nombre;
48     }
49     public double getPrecioProducto()
50     {
51         return this.precioReferenciaKg;
52     }
53     public double getPesoProducto()
54     {
55         return this.pesoToneladas;
56     }
57     public boolean getEsPerecederoProducto()
58     {
59         return this.esPerecedero;
60     }
61     public Date getFechaCompra(){
62         return this.fechaCompra;
63     }
64
65     /**Metodos set
66     */
67     public void setFechaCompra(Date fechaCompra){
68         this.fechaCompra = fechaCompra;
69     }
70     /**Convierte las toneladas a kilogramos
71     */
72
73     public double getPesoEnKg(){
74         pesoEnKg = pesoToneladas * 1000;
75         return this.pesoEnKg;
76     }
77     @Override
78     public String toString(){
79         return "Producto: " + nombre + ", precio: " + precioReferenciaKg;
80     }
81 }

```

La clase Producto representa un artículo gestionado en la cooperativa. Tiene atributos como el nombre, precio de referencia por kilogramo, peso en toneladas, indicador de si es perecedero y la fecha de compra. También contiene una lista de productos representada por un ArrayList.

El constructor de la clase permite inicializar los atributos del producto. Se pueden proporcionar el nombre, precio, peso y si es perecedero. También hay un constructor adicional para casos donde solo se proporciona el precio y el peso.

La clase Producto proporciona métodos de acceso (get) para obtener el nombre, precio, peso y si es perecedero del producto. Además, hay un método getFechaCompra para obtener la fecha de compra.

La clase también incluye métodos de modificación (set) para establecer la fecha de compra.

El método getPesoEnKg convierte el peso en toneladas a kilogramos.

Además, se ha implementado el método toString para devolver una representación en cadena del producto, que incluye el nombre y el precio.

```

1
2
3     public class Coste extends Producto {
4
5         // variables de instancia
6         private static int distanciaKm;
7         private double pesoPedido;
8         private double precioArticulo;
9         private boolean esPerecedero;
10
11         public Coste() {
12             super("Arroz", distanciaKm, 0, true); // String double, double, boolean

```

```

13         this.precioArticulo = precioArticulo;
14         this.esPerecedero = esPerecedero;
15         this.distanciaKm = distanciaKm;
16         this.pesoPedido = pesoPedido;
17     }
18
19
20     public int getDistanciaKm() {
21         return this.distanciaKm;
22     }
23
24     public String PagarProveedor(double precioArticulo, double pesoArticulo) {
25
26         double tonelada = 1000;
27         pesoPedido = (pesoArticulo * tonelada);
28         double pagar = (precioArticulo * pesoPedido);
29         String pagarString = Double.toString(pagar);
30
31         return "Pagar al proveedor: " + pagarString;
32     }
33
34     public String calcularCosteLogistica() {
35
36         return "Coste Logistica clase coste:";
37     }
38
39 }

```

La clase Coste hereda de la clase Producto y se encarga de calcular el costo a pagar al proveedor y el costo de la logística.

La clase contiene variables de instancia como la distancia en kilómetros, el peso del pedido, el precio del artículo y un indicador de si es perecedero.

El constructor de la clase inicializa los atributos con valores predeterminados, utilizando el constructor de la clase base (Producto) y asignando valores a las variables de instancia.

La clase proporciona un método getDistanciaKm para obtener la distancia en kilómetros.

La clase Coste se encarga de calcular los costos relacionados con el proveedor y la logística, utilizando información como la distancia, el peso y el precio del artículo. Hereda de la clase Producto y extiende su funcionalidad para proporcionar cálculos específicos de costos.

```

1
2  /**
3   * Clase Producto que contiene las características necesarias de cada producto para
4     que pueda ser gestionado en la cooperativa.
5   *
6   * @author (Angela Alexandra Guzman Garcia)
7   * @version (001)
8   */
9   public class Logistica extends Coste
10  {
11      // instance variables - replace the example below with your own
12      private int distanciaKm;
13      private boolean esPerecedero;
14      private double pesoPedido;
15      public double precioArticulo;
16
17      /**
18       * Constructor for objects of class Logistica
19       */
20      public Logistica(int distanciaKm)
21      {
22          // inicializa variables de instancia
23          super();
24          this.distanciaKm = distanciaKm;
25          this.esPerecedero = esPerecedero;

```

```

26         this.pesoPedido = pesoPedido;
27     }
28
29     public int getDistancia()
30     {
31         return this.distanciaKm;
32     }
33     public void setDistancia(int distanciaKm) {
34         this.distanciaKm = distanciaKm;
35     }
36     public boolean getPerecedero()
37     {
38         return this.esPerecedero;
39     }
40
41     public void setEsPerecedero(boolean esPerecedero) {
42         this.esPerecedero = esPerecedero;
43     }
44
45     public String Transportar(){
46         return "transportar de logistica";
47     }
48 }

```

la clase Logistica se utiliza para gestionar la logística de un producto en la cooperativa. Hereda funcionalidad de la clase Coste y proporciona métodos y atributos específicos como la distancia, la perecibilidad y la acción de transportar.

La clase tiene variables de instancia como la distancia en kilómetros, un indicador de si el producto es perecedero, el peso del pedido y el precio del artículo. El constructor de la clase recibe la distancia en kilómetros y asigna los valores a las variables de instancia.

De la misma manera la clase proporciona métodos para obtener y establecer la distancia, y para obtener y establecer si el producto es perecedero y el método Transportar devuelve una cadena que indica la acción de transportar asociada a la logística.

```

1  /**
2   * Clase PequenaLogistica que hereda los metodos Transportar y
3   * calcularCosteLogistica de la
4   * clase logistica, aplica polimorfismosobreescribiendo los metodos para ajustar a
5   * las características
6   * particulares de la gran logistica.
7   * @author (Angela Alexandra Guzman Garcia)
8   * @version (001)
9   */
10 public class PequenaLogistica extends Logistica
11 {
12     // variables de instancia
13     private int distanciaKm;
14     private double costeFijoKm = 0.3;
15     private double coste;
16
17     /** Constructor inicializa las variables
18     */
19     public PequenaLogistica( int distanciaKm)
20     {
21         // inicializa variables
22         super(distanciaKm);
23         this.distanciaKm = distanciaKm;
24     }
25
26     @Override
27     public String Transportar(){
28         if(distanciaKm <= 100 ){
29             System.out.println("-----");
30             System.out.println("Enviando producto a traves de pequena logistica.");
31             System.out.println("-----");

```

```

30     }
31     String distanciaString = Integer.toString(distanciaKm);
32     return "Distancia a recorrer: " + distanciaString + " Kilometros.";
33 }
34 /**Se sobrescribe el metodo calcularCosteLogistica para ajustar a Gran Logistica
35 */
36 @Override
37 public String calcularCosteLogistica(){
38     costeFijoKm = 3;
39     coste = costeFijoKm * distanciaKm;
40     return "Coste Pequenia Logistica: " + coste + " Euros.";
41 }
42 int totalPLogistica = 6000;
43 public void TotalLogistica(){
44     System.out.println("-----");
45     System.out.println("El precio total de la pequena logistica el ultimo anio
46         fue);
47 }

```

la clase PequenaLogistica representa la logística de la cooperativa para distancias cortas. Aplica el polimorfismo al sobrescribir métodos de la clase base Logistica para adaptarlos a las características particulares de la pequeña logística. Proporciona métodos y atributos específicos relacionados con el cálculo de costes y el envío de productos a través de la pequeña logística. El constructor de la clase recibe la distancia en kilómetros y asigna el valor a la variable de instancia correspondiente.

El método Transportar sobrescrito imprime un mensaje indicando que el producto se enviará a través de la pequeña logística. Luego devuelve una cadena que muestra la distancia a recorrer en kilómetros.

El método calcularCosteLogistica tambien sobrescrito establece el coste fijo por kilómetro para la pequeña logística y calcula el coste total basado en la distancia. Luego devuelve una cadena que muestra el coste de la pequeña logística.

```

1
2  /**
3   * Clase GranLogistica que contiene los metodos Transportar y
4   * calcularCosteLogistica
5   * sobreescritos con polimorfismo para ajustar a las caracteristicas
6   * particulares
7   * de la gran logistica
8   *
9   * @author (Angela Alexandra Guzman Garcia)
10  * @version (001)
11  */
12 public class GranLogistica extends Logistica // Hereda de la clase Logistica
13 {
14     // variables de instancia
15     private int distanciaKm;
16     private boolean esPerecedero;
17     private int tramos;
18     private double costeProducto;
19     private String ciudad;
20     private int restoKm;
21     private double pesoPedido;
22
23
24     /**
25      * Constructor inicializa las variables
26      */
27     public GranLogistica(String ciudad, int distanciaKm) {
28         // initialise instance variables
29         super(distanciaKm);
30         this.distanciaKm = distanciaKm;
31         this.esPerecedero = getPerecedero();
32         this.costeProducto = getPrecioProducto();
33         this.ciudad = ciudad;
34         this.pesoPedido = pesoPedido;

```

```

35     }
36
37
38     /**
39     * Se sobrescribe el metodo transportar para ajustar a Gran Logistica
40     */
41     @Override
42     public String Transportar() {
43         tramos = distanciaKm / 50;
44         restoKm = distanciaKm % 50;
45         if (esPerecedero == true && distanciaKm > 100) {
46             System.out.println("-----");
47             System.out.println("Enviando producto a " + ciudad + ".\n");
48             System.out.println("-----");
49             System.out.println(
50                 "Distancia a recorrer gran logistica: " + distanciaKm + "Km, en
                    " + tramos + " tramos de 50Km.");
51             System.out.println("Distancia a recorrer pequena logistica: " +
                restoKm + "Km.");
52         }
53         return "Envio a " + ciudad + " a " + distanciaKm + " Km, en " + tramos + "
            tramos de 50Km y " + restoKm
54             + "Km en pequena logistica";
55     }
56
57     /**
58     * Se sobrescribe el metodo calcularCosteLogistica para ajustar a Gran Logistica
59     */
60     // @Override
61     public String calcularCosteLogistica(double pesoArticulo, int distanciaKm,
        double costeProducto) {
62         int tramos = (distanciaKm / 50);
63         double costeTotal = 0;
64         for (int i = 0; i < tramos; i++) {
65             double costeTramo = 0.5 * costeProducto * pesoArticulo;
66             costeTotal += costeTramo;
67         }
68         String costeString = Double.toString(costeTotal);
69         return "El coste de la Gran logistica es de: " + costeString + " Euros.";
70     }
71 }

```

la clase GranLogistica representa la logística de la cooperativa para distancias largas. Utiliza la herencia y el polimorfismo para ajustar los métodos de la clase base Logistica a las características específicas de la gran logística. Proporciona métodos y atributos relacionados con el cálculo de costes, la distribución de kilómetros y el envío de productos a través de la gran logística.

El constructor de la clase recibe la ciudad y la distancia en kilómetros, y asigna los valores correspondientes a las variables de instancia.

El método Transportar sobrescrito calcula el número de tramos de 50 kilómetros y el resto de kilómetros que se enviarán a través de la pequeña logística. Si el producto es perecedero y la distancia es mayor a 100 kilómetros, muestra un mensaje indicando la ciudad de destino y la distancia a recorrer tanto en la gran logística como en la pequeña logística. Luego devuelve una cadena que indica el envío a la ciudad y la distribución de los kilómetros.

El método calcularCosteLogistica sobrescrito calcula el coste total de la gran logística basado en el peso del artículo, la distancia en kilómetros y el coste del producto. Divide la distancia en tramos de 50 kilómetros y calcula el coste de cada tramo. Luego devuelve una cadena que muestra el coste total de la gran logística.

```

1
2  /**
3  * Clase GestionProductos que realiza las acciones de calcular los importes a cada
    tipo de cliente.
4  * @author (Angela Alexandra Guzman Garcia)
5  * @version (001)
6  */
7  public class GestionProductos

```

```

8 {
9     // variables de instancia
10    private double precioRefKg;
11    private double precioClienteF;
12    private double beneficiosCF;
13    private double precioDistribuidor;
14    private double beneficiosDistribuidor;
15    private double kilos;
16    private double totalPagar;
17
18
19    /**
20     * Constructor para objetos de la clase GestionProductos
21     */
22    public GestionProductos()
23    {
24        // inicializacion de variables de constructor
25
26        Producto miProducto = new Producto ("Nombre producto", precioRefKg , 0.0, false
27        );
28        this.precioRefKg = miProducto.getPrecioProducto();
29        this.kilos = miProducto.getPesoEnKg();
30
31    }
32
33    /** Funcion que aumenta el 15% al precio del cliente final segun el precio de
34        referencia por kilogramo del producto
35        */
36    public double CobrarACliente(double precioRefKg)
37    {
38        // put your code here
39        beneficiosCF = precioRefKg * 15 / 100;
40        precioClienteF = precioRefKg + beneficiosCF;
41        return precioClienteF;
42    }
43
44    /** Funcion que aumenta el 5% al precio del distribuidor segun el precio de
45        referencia del producto
46        */
47    public double CobrarDistribuidor(double precioRefKg)
48    {
49        beneficiosDistribuidor = precioRefKg * 5 / 100;
50        precioDistribuidor = precioRefKg + beneficiosDistribuidor;
51        return precioDistribuidor;
52    }
53 }

```

La clase GestionProductos se encarga de realizar acciones relacionadas con el cálculo de importes para diferentes tipos de clientes. Tiene variables de instancia que almacenan precios, beneficios y cantidades de productos. El constructor inicializa estas variables a través de la creación de un objeto de la clase Producto.

```

1  import java.util.List;
2  import java.util.ArrayList;
3  import java.util.Scanner;
4
5  /**
6   * Clase GestionPedidos gestiona los productos
7   * disponibles en la cooperativa para comprar.
8   * @author (Angela Alexandra Guzman Garcia)
9   * @version (001)
10   */
11  public class GestionPedidos
12  {
13      /**El metodo ListaProductos, crea elementos tipo Producto y luego los

```

```

14         * agrega a una lista, para finalmente mostrar la lista al usuario.
15         */
16         String agregarMas = "";
17         String prodIngresado = "";
18         //Declara una lista de productos
19         List<Producto> listaProductos = new ArrayList<>();
20         //Crea nueva lista de productos ingresados por el usuario.
21         List<String> listCesta = new ArrayList<>();
22
23     public void ListaProductos(){
24         //Crea elementos tipo Producto
25         Producto manzana = new Producto("Manzana", 3.4, 2, true);
26         Producto pera = new Producto("Pera", 2.1, 1.5, true);
27         Producto algodón = new Producto("Algodón", 4.3, 6, false);
28         Producto pimienta = new Producto("Pimiento", 2.2, 3, true);
29         Producto arroz = new Producto("Arroz", 1.4, 2.2, true);
30         Producto cebolla = new Producto("Cebolla", 2.4, 2.6, true);
31         Producto brocoli = new Producto("Brocoli", 1.4, 2.5, true);
32
33         //agrega elementos a la lista
34         listaProductos.add(manzana);
35         listaProductos.add(pera);
36         listaProductos.add(algodón);
37         listaProductos.add(pimiento);
38         listaProductos.add(arroz);
39         listaProductos.add(cebolla);
40         listaProductos.add(brocoli);
41
42         //Imprime la lista de productos disponibles
43         for(Producto producto : listaProductos){
44             System.out.println(producto.toString());
45         }
46     }
47     public void Cesta(){
48         do{
49             Scanner sc = new Scanner(System.in);
50             System.out.println("Indique el nombre del producto a comprar");
51             //Lee el producto aniadido por consola a la cesta
52             String prodIngresado = sc.nextLine();
53             //Aniade el producto a la cesta
54             System.out.println("Desea agregar otro producto? si/no");
55             agregarMas = sc.nextLine();
56             //Verifica que la respuesta sea correcta
57             while(!agregarMas.equals("si") && !agregarMas.equals("no")){
58                 System.out.println("Valor no valido, intente nuevamente: ");
59                 agregarMas = sc.nextLine();
60             }
61
62             listCesta.add(prodIngresado);
63             //Recorre la lista y verifica que el producto ingresado este en la
64             lista para luego removerlo
65             for (int i= 0; i < listaProductos.size(); i++){
66                 if(listaProductos.get(i).getNombreProducto().equalsIgnoreCase(
67                     prodIngresado)){
68                     listaProductos.remove(i);
69                     break;
70                 }
71             }
72             }while(!agregarMas.equals("no")) ;
73             //Imprime los productos de la cesta
74             System.out.println("-----");
75             System.out.println("Productos en la cesta.");
76             System.out.println("-----");
77             for(String producto : listCesta){
78                 System.out.println(producto);
79             }

```



```

78         //Imprime de nuevo los productos disponibles
79         System.out.println("-----");
80         System.out.println("Productos disponibles.");
81         System.out.println("-----");
82         for(int i = 0; i < listaProductos.size(); i++){
83             System.out.println(listaProductos.get(i).getNombreProducto());
84         }
85     }
86 }

```

La clase GestionPedidos se encarga de gestionar los productos disponibles en la cooperativa para su compra. Tiene dos métodos principales: ListaProductos y Cesta.

El método ListaProductos crea varios objetos de la clase Producto con diferentes características y los agrega a una lista llamada listaProductos. Luego, muestra por consola la lista de productos disponibles.

El método Cesta permite al usuario agregar productos a su cesta de compra. Solicita al usuario el nombre del producto a comprar, verifica si desea agregar más productos y registra los productos ingresados en una lista llamada listCesta. También elimina los productos seleccionados de la lista de productos disponibles. Finalmente, muestra por consola los productos en la cesta y los productos disponibles actualizados.

```

1  import java.util.ArrayList;
2  import java.util.Scanner;
3
4  /**
5   * Clase Pedido que pide al proveedor que ingrese los productos
6   * y los pone en una cesta de productos seleccionados.
7   * @author (Angela Alexandra Guzman Garcia)
8   * @version (001)
9   */
10 public class Pedido
11 {
12     //Crea un array de productos tipo producto
13     private ArrayList<Producto> listaProductos;
14     //Declara el constructor
15     public Pedido(ArrayList<Producto> listaProductos)
16     {
17         this.listaProductos = listaProductos;
18     }
19
20
21
22     //Metodo que agregados un producto a la lista
23     public void ListaProductosAdd(){
24         Scanner sc = new Scanner(System.in);
25         String agregar = "si";
26
27         do{
28             System.out.println("-----");
29             System.out.println("Ingrese el producto a agregar");
30             System.out.println("-----");
31
32             System.out.println("Ingrese el nombre del producto: ");
33             String nombre = sc.nextLine();
34             System.out.println("Ingrese el precio del producto: ");
35             while(!sc.hasNextDouble()){
36                 System.out.println("El valor ingresado no es valido, intente nuevamente: ");
37                 sc.next();
38             }
39             double precio = sc.nextDouble();
40             System.out.println("Ingrese cantidad (toneladas) del producto: ");
41             double cantidad = sc.nextDouble();
42             System.out.println("Es perecedero true/false:");
43             boolean perecedero = sc.nextBoolean();
44             System.out.println("-----");
45             System.out.println("Desea agregar mas productos: si/no");

```

```

46         System.out.println("-----");
47         String teclado = sc.nextLine();
48
49         Producto producto1 = new Producto(nombre, precio, cantidad, perecedero)
50         ;
51         this.listaProductos.add(producto1);
52
53         agregar = sc.nextLine();
54     }while(agregar.equalsIgnoreCase("si"));
55 }
56
57     public void ListaPedido(){
58     for (Producto producto : listaProductos){
59         System.out.println(producto.getNombreProducto());
60     }
61 }
62 }

```

La clase Pedido se utiliza para solicitar al proveedor que ingrese los productos y almacenarlos en una lista de productos seleccionados. Tiene dos métodos principales: ListaProductosAdd y ListaPedido.

El método ListaProductosAdd permite al proveedor ingresar los detalles de un producto, como el nombre, el precio, la cantidad y si es perecedero. Luego, crea un objeto de la clase Producto con los detalles ingresados y lo agrega a la lista listaProductos. El proveedor puede continuar ingresando más productos hasta que decida no agregar más.

El método ListaPedido muestra por consola los nombres de los productos almacenados en la lista listaProductos.

```

1
2     /**
3     * Clase Productor, contiene los detalles de cada producto y los metodos set y get.
4     * @author (Angela Alexandra Guzman Garcia)
5     * @version (001)
6     */
7     public class Productor
8     {
9         // Variables
10        private String nombre;
11        private String producto;
12        private int hectareas;
13        private int toneladas;
14
15        public Productor(String nombre, String producto, int hectareas, int toneladas) {
16            this.nombre = nombre;
17            this.producto = producto;
18            this.hectareas = hectareas;
19            this.toneladas = toneladas;
20        }
21
22        public String getNombre() {
23            return nombre;
24        }
25
26        public void setNombre(String nombre) {
27            this.nombre = nombre;
28        }
29
30        public String getProducto() {
31            return producto;
32        }
33
34        public void setProducto(String producto) {
35            this.producto = producto;
36        }
37

```

```

38     public int getHectareas() {
39         return hectareas;
40     }
41
42     public void setHectareas(int hectareas) {
43         this.hectareas = hectareas;
44     }
45
46     public int getToneladas() {
47         return toneladas;
48     }
49
50     public void setToneladas(int toneladas) {
51         this.toneladas = toneladas;
52     }
53 }

```

La clase Productor contiene los detalles de cada producto, como el nombre del productor, el nombre del producto, la cantidad de hectáreas y la cantidad de toneladas producidas.

Tiene un constructor que toma los valores de los atributos y los asigna a las variables correspondientes.

Además, la clase proporciona métodos getter y setter para acceder y modificar los valores de los atributos.

```

1  import java.util.ArrayList;
2  import java.util.Calendar;
3  import java.util.GregorianCalendar;
4
5  /**
6   * Clase PequenoProductor que realiza acciones con pequenos productores
7   * @author (Angela Alexandra Guzman Garcia)
8   * @version (001)
9   */
10 public class PequenoProductor extends Productor
11 {
12     // Variables de instancia
13     private int anio;
14     private int ultimoDigitoAnio;
15     public String[] arrayPPProductores;
16
17     /** Se llama al constructor de la super clase Productor a traves de la palabra
18         reservada super
19     */
20     public PequenoProductor(String nombre, String producto, int hectareas, int
21         toneladas) {
22         super(nombre, producto, hectareas, toneladas);
23     }
24
25     /** metodo que calcula el ultimo digito del anio
26     */
27     public Integer getYear() {
28         Calendar fecha = new GregorianCalendar();
29         anio = fecha.get(Calendar.YEAR);
30         // Calcula el ultimo digito del anio
31         ultimoDigitoAnio = anio % 10;
32         return ultimoDigitoAnio;
33     }
34
35     public void MostrarPequeProductores(){
36         ArrayList<Productor> listaPPProductores = new ArrayList<Productor>();
37         listaPPProductores.add(new Productor("Maria", "Cafe", 10, 100));
38         listaPPProductores.add(new Productor("Juana", "Cacao", 15, 200));
39         listaPPProductores.add(new Productor("Antonia", "Platano", 20, 300));
40         listaPPProductores.add(new Productor("Laura", "Papa", 25, 400));
41         listaPPProductores.add(new Productor("Sofia", "Frijol", 30, 500));

```

```

42         // Agregar mas elementos a la lista
43         listaPProductores.add(new Productor("Pedro", "Cafe", 12, 120));
44         listaPProductores.add(new Productor("Manuel", "Cacao", 17, 220));
45         listaPProductores.add(new Productor("Lucia", "Platano", 22, 320));
46
47         // Acceder a un elemento de la lista
48         Productor primerProductor = listaPProductores.get(0);
49         String nombrePrimerProductor = primerProductor.getNombre();
50         String productoPrimerProductor = primerProductor.getProducto();
51         int hectareasPrimerProductor = primerProductor.getHectareas();
52         int toneladasPrimerProductor = primerProductor.getToneladas();
53
54     }
55 }

```

La clase `PequenoProductor` es una subclase de la clase `Productor` y se utiliza para realizar acciones específicas relacionadas con los pequeños productores.

Tiene dos variables de instancia adicionales: `anio` y `ultimoDigitoAnio`. La variable `anio` almacena el año actual obtenido del calendario, mientras que `ultimoDigitoAnio` calcula el último dígito del año actual.

El constructor de la clase utiliza la palabra clave `super` para llamar al constructor de la superclase `Productor` y pasar los valores correspondientes.

La clase también contiene un método `getYear` que obtiene el año actual y calcula el último dígito del año. El resultado se devuelve como un entero.

Además, la clase tiene un método `MostrarPequeProductores` que crea una lista de pequeños productores, agrega elementos a la lista y accede a los detalles de un productor específico de la lista.

```

1  /**
2   * Clase GranProductor que hereda de productor y cuenta con el constructor de la
3   * clase para ser heredada
4   * por otras clases.
5   * @author (Angela Alexandra Guzman Garcia)
6   * @version (001)
7   */
8  public class GranProductor extends Productor
9  {
10     /**
11      * Constructor for objects of class GranProductor
12      */
13     public GranProductor(String nombre, String producto, int hectareas, int
14         toneladas, String categoria) {
15         super(nombre, producto, hectareas, toneladas);
16     }
17 }

```

La clase `GranProductor` es una subclase de la clase `Productor` y se utiliza para representar a los grandes productores. Tiene un constructor que toma los mismos parámetros que el constructor de la clase `Productor`, además de un parámetro adicional llamado `categoria`, que representa la categoría del gran productor.

El constructor de la clase utiliza la palabra clave `super` para llamar al constructor de la superclase `Productor` y pasar los valores correspondientes.

```

1  /**
2   * Clase ProductorFederado que extiende de productor y contiene el constructor de la
3   * clase
4   * para que pueda ser heredado por otras clses.
5   * @author (Angela Alexandra Guzman Garcia)
6   * @version (001)
7   */
8  public class ProductorFederado extends Productor
9  {
10
11
12     /**

```

```

13      * Constructor for objects of class ProductorFederado
14      */
15      public ProductorFederado(String nombre, String producto, int hectareas, int
16          toneladas, String categoria) {
17          super(nombre, producto, hectareas, toneladas);
18      }
19  }

```

La clase ProductorFederado es una subclase de la clase Productor y se utiliza para representar a los productores federados. Tiene un constructor que toma los mismos parámetros que el constructor de la clase Productor, además de un parámetro adicional llamado categoria, que representa la categoría del productor federado.

```

1      import java.security.KeyStore.Entry;
2      import java.util.Hashtable;
3
4      /**
5       * Clase Informes que contiene el metodo informe de cantidad vendida para
6       * mostrarle *al usuario administrador
7       * @author (Angela Alexandra Guzman Garcia)
8       * @version (001)
9       */
10     public class InformeVentas {
11         //Diccionario
12         Hashtable<String, Integer> cantidadVendida = new Hashtable<String, Integer>();
13
14         //Añade productos clave valor al diccionario cantidadVendida
15         public void informeCantidadVendida() {
16             cantidadVendida.put("Manzana", 220);
17             cantidadVendida.put("Pera", 20);
18             cantidadVendida.put("Algodon", 120);
19             cantidadVendida.put("Pimiento", 250);
20             cantidadVendida.put("Arroz", 267);
21             cantidadVendida.put("Cebolla", 280);
22             cantidadVendida.put("Brocoli", 180);
23
24             //Muestra el listado de productos entoneladas vendidos en el ultimo año
25             System.out.println("Cantidad de productos en toneladas vendidas el ultimo
26                 año: \n");
27             for (java.util.Map.Entry<String, Integer> entry : cantidadVendida.entrySet
28                 ()) {
29                 System.out.println(entry.getKey() + ": " + entry.getValue() + "
30                     toneladas.");
31             }
32         }
33     }

```

La clase InformeVentas representa un informe de cantidad vendida de productos. Utiliza un diccionario llamado cantidadVendida para almacenar la cantidad vendida de cada producto.

El método informeCantidadVendida agrega los productos y sus respectivas cantidades vendidas al diccionario. Luego, recorre el diccionario y muestra el listado de productos y las cantidades vendidas en toneladas en el último año.

```

28      import java.util.Hashtable;
29
30      /**
31       * Clase InformeEmpresa que contiene un metodo que muestra al usuario
32       * administrador informacion sobre las empresas de transporte.
33       * @author (Angela Alexandra Guzman Garcia)
34       * @version (001)
35       */
36      public class InformeEmpresa{
37          //Importes obtenidos por cada una de las empresas de logistica.
38          public void infoEmpresas(){

```

```

39 //Diccionario
40 Hashtable<String, Integer> importeEmpresas= new Hashtable<String, Integer>
    >();
41
42 importeEmpresas.put("Logis S.A", 4556);
43 importeEmpresas.put("Transportes TM", 8034);
44 importeEmpresas.put("DHL", 4435);
45
46
47 //Muestra el listado de productos entoneladas vendidos en el ultimo anio
48 System.out.println("* Importe obtenido segun empresa: \n");
49 for (java.util.Map.Entry<String, Integer> entry : importeEmpresas.entrySet
    ()) {
50     System.out.println(entry.getKey() + ": " + entry.getValue() + " euros."
        );
51 }
52 }
53 }

```

La clase InformeEmpresa proporciona información sobre las empresas de transporte. Utiliza un diccionario llamado importeEmpresas para almacenar los importes obtenidos por cada una de las empresas de logística.

```

1 import java.util.Hashtable;
2 /**
3  * Clase InformeProductor que contiene un metodo que muestra al usuario
4  * administrador informacion sobre el rendimiento de la cooperativa.
5  * @author (Angela Alexandra Guzman Garcia)
6  * @version (001)
7  */
8
9 public class InformeProductor {
10     //Diccionario
11     Hashtable<String, Integer> importeProductor = new Hashtable<String, Integer>();
12
13     //Añade productos clave valor al diccionario cantidadVendida
14     public void infoRendimiento() {
15         //mportes obtenidos por cada uno de los productores (desglosados por
            productos)
16         importeProductor.put("Juan", 3456);
17         importeProductor.put("Maria", 2034);
18         importeProductor.put("Rosa", 9435);
19         importeProductor.put("Antonio", 2340);
20         importeProductor.put("Jose", 2647);
21         importeProductor.put("Luis", 2807);
22         importeProductor.put("Carmen", 1480);
23
24         //Muestra el listado de productos entoneladas vendidos en el ultimo anio
25         System.out.println("* Importe obtenido segun productor: \n");
26         for (java.util.Map.Entry<String, Integer> entry : importeProductor.entrySet
            ()) {
27             System.out.println(entry.getKey() + ": " + entry.getValue() + " euros."
                );
28         }
29     }
30 }
31 }

```

La clase InformeProductor contiene un método que muestra al usuario administrador información sobre el rendimiento de la cooperativa. Utiliza un diccionario llamado importeProductor para almacenar los importes obtenidos por cada uno de los productores.

El método infoRendimiento agrega los productores y sus respectivos importes al diccionario. Luego, recorre el diccionario y muestra el listado de productores y los importes obtenidos en euros.

```

1 import java.util.Hashtable;
2 /**

```

```

3      * Clase InformeProductos que contiene un metodo que muestra al usuario
4      * administrador informacion sobre las empresas de transporte.
5      * @author (Angela Alexandra Guzman Garcia)
6      * @version (001)
7      */
8      public class InformeProductos {
9          //Diccionario
10         Hashtable<String, Integer> beneficiosProducto = new Hashtable<String, Integer>
11             >();
12
13         //Aniade productos clave valor al diccionario
14         public void infoRendimiento() {
15             //mportres obtenidos por cada uno de los productores (desglosados por
16             productos)
17             beneficiosProducto.put("Manzana", 56);
18             beneficiosProducto.put("Pera", 34);
19             beneficiosProducto.put("Algodon", 95);
20             beneficiosProducto.put("Pimiento", 20);
21             beneficiosProducto.put("Arroz", 27);
22             beneficiosProducto.put("Cebolla", 07);
23             beneficiosProducto.put("Brocoli", 10);
24
25             //Muestra el listado
26             System.out.println("* Beneficios por producto: \n");
27             for (java.util.Map.Entry<String, Integer> entry : beneficiosProducto.
28                 entrySet()) {
29                 System.out.println(entry.getKey() + ": " + entry.getValue() + " euros."
30                     );
31             }
32         }
33     }

```

La clase InformeProductos contiene un método que muestra al usuario administrador información sobre los beneficios de cada producto. Utiliza un diccionario (Hashtable) llamado beneficiosProducto para almacenar los beneficios de cada producto.

El método infoRendimiento agrega los productos y sus respectivos beneficios al diccionario. Luego, recorre el diccionario y muestra el listado de productos y los beneficios obtenidos en euros.

```

1
2      /**
3      * Clase ImprimirMenu que contiene los diferentes menus que se le presentan
4      * al usuario al momento de ejecutar la aplicacion.
5      * @author (Angela Alexandra Guzman Garcia)
6      * @version (001)
7      */
8
9      public class ImprimirMenu
10     {
11         public void MenuBienvenido(){
12             System.out.println("=====");
13             System.out.println("      Bienvenido a la coooperativa.  \n");
14             System.out.println("-----");
15             System.out.println("-----MENU PRINCIPAL-----");
16             System.out.println("Elija la opcion que desee...");
17             System.out.println("1. Soy proveedor, vender mis productos ");
18             System.out.println("2. Soy cliente, comprar productos");
19             System.out.println("3. Soy administrador, generar informes");
20             System.out.println("4. Salir");
21             System.out.println("-----");
22         }
23         public void MenuVender()
24         {
25             // put your code here

```

```

26         System.out.println("-----");
27         System.out.println("Bienvenido asociado \n");
28         System.out.println("-----");
29
30     }
31     public void MenuComprar()
32     {
33         // put your code here
34         System.out.println("Lista de productos disponibles \n");
35
36     }
37     public void MenuInformes()
38     {
39         // put your code here
40         System.out.println("-----");
41         System.out.println("Bienvenido administrador\n");
42         System.out.println("-----");
43         System.out.println("Elija la opcion que desee...");
44         System.out.println("1. Generar informe de productos ");
45         System.out.println("2. Generar informe de ventas");
46         System.out.println("3. Generar informe de rendimiento");
47         System.out.println("-----");
48
49     }
50 }

```

La clase ImprimirMenu contiene métodos para imprimir diferentes menús que se le presentan al usuario al ejecutar la aplicación.

El método MenuBienvenido imprime el menú principal de la cooperativa, donde el usuario puede elegir entre ser proveedor, cliente o administrador, o salir de la aplicación.

El MenuVender imprime el menú para los proveedores, donde se les da la bienvenida y pueden realizar acciones relacionadas con la venta de sus productos.

El MenuComprar imprime el menú para los clientes, mostrando la lista de productos disponibles para comprar.

En el MenuInformes imprime el menú para el administrador, donde se les da la bienvenida y pueden generar diferentes informes relacionados con productos, ventas y rendimiento.

```

28 import java.util.Hashtable;
29 /**
30  * Clase InformeProductos que contiene un metodo que muestra al usuario
31  * administrador informacion sobre las empresas de transporte.
32  * @author (Angela Alexandra Guzman Garcia)
33  * @version (001)
34  */
35 public class InformeProductos {
36     //Diccionario
37     Hashtable<String, Integer> beneficiosProducto = new Hashtable<String, Integer>
38         >();
39
40     //Añade productos clave valor al diccionario
41     public void infoRendimiento() {
42         //mportos obtenidos por cada uno de los productores (desglosados por
43         productos)
44         beneficiosProducto.put("Manzana", 56);
45         beneficiosProducto.put("Pera", 34);
46         beneficiosProducto.put("Algodon", 95);
47         beneficiosProducto.put("Pimiento", 20);
48         beneficiosProducto.put("Arroz", 27);
49         beneficiosProducto.put("Cebolla", 07);
50         beneficiosProducto.put("Brocoli", 10);
51
52         //Muestra el listado
53         System.out.println("* Beneficios por producto: \n");

```



```
54         for (java.util.Map.Entry<String, Integer> entry : beneficiosProducto.  
55             entrySet()) {  
56             System.out.println(entry.getKey() + ": " + entry.getValue() + " euros."  
57                 );  
58         }  
    }
```

Finalmente, la clase InformeProductos contiene un método llamado infoRendimiento que muestra al usuario administrador información sobre los beneficios de cada producto en euros.

La clase utiliza una Hashtable llamada beneficiosProducto para almacenar los beneficios de cada producto, donde la clave es el nombre del producto y el valor es la cantidad de beneficios en euros.

El método infoRendimiento añade los productos y sus beneficios al diccionario beneficiosProducto y luego muestra el listado de productos y sus beneficios en euros utilizando un bucle for-each sobre las entradas del diccionario.