INTRODUCCIÓN MEMORIA

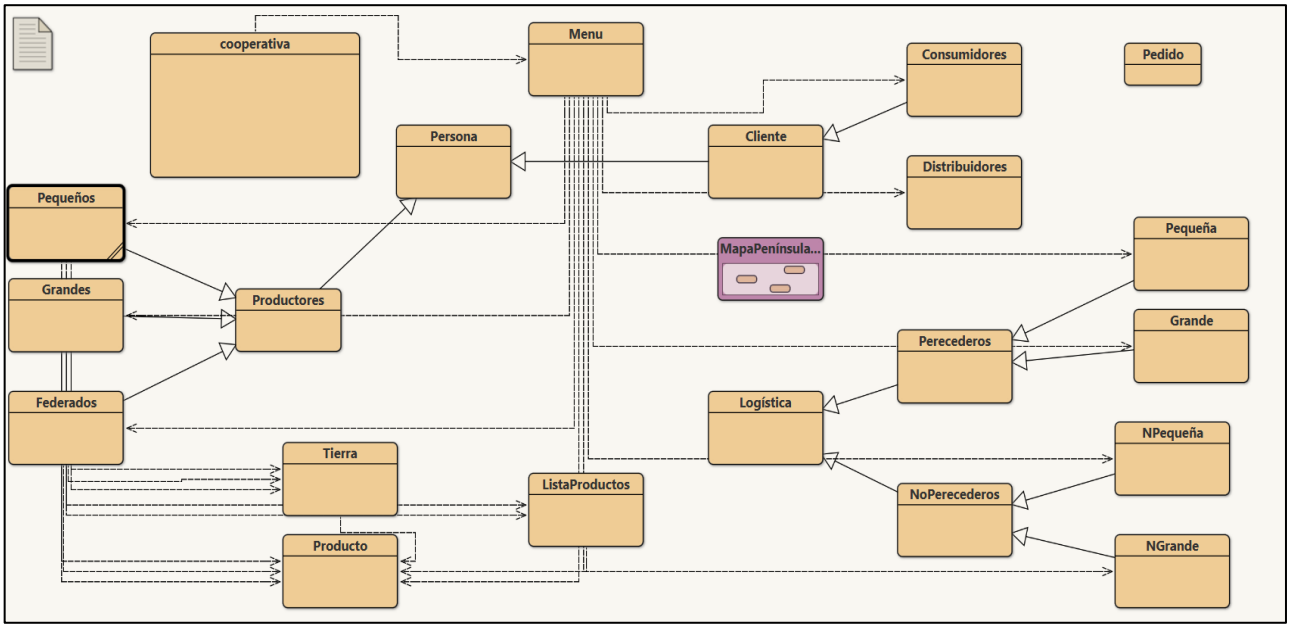
En la memoria trataré de explicar cada uno de los elementos aplicados y el porqué de forma resumida. Lo que pretendo es abarcar el mayor número de funciones y tareas que se piden en el enunciado, de la forma en la que he aprendido a la vez que estudiaba el libro de la asignatura de “Programación Orientada a Objetos”, por lo tanto, la programación de este programa se basa en el uso de clases, objetos y herencia. Algunas funciones no he llegado a programarlas o las he programado de forma muy básica debido a que tampoco hay mucho tiempo para hacer algo tan grande, si tenemos que ir aprendiendo a la vez la asignatura y compatibilizarlas con otras. Pero bueno, intentaré explicar todo lo mejor posible y disculpa por la falta de claridad tal vez, por la existencia de tantísimo código.

Los apartados más importantes sobre los que hablaré son:
1. Diagrama de Clases.
2. Explicación de las Clases.
    2.1. Clase Productor.
    2.2. Clase Logística.
    2.3. Clase Cliente.
3. La clase Menú y el Main.
4. Otros apuntes.

---

1. DIAGRAMA DE CLASES

Este es el diagrama de clases, la clase pedidos no conecta con ninguna porque todavía no está programada, pero si siguiese con el programa la usaría para gestionar todos los pedidos y costes de los pedidos de los clientes. Aquí va.



2. EXPLICACIÓN DE LAS CLASES

2.1. CLASE PRODUCTOR

La clase “Productor” y sus hijos son las clases que completan los requisitos del enunciado dentro de “Productores”. Vamos a empezar por la clase “padre”, la clase “Persona”.

Clase Persona:

```
public abstract class Persona
{   //campos de la clase persona
    private String nombre,dni;
    private int dianacimiento,mesnacimiento,añonacimiento;

public Persona(String nombre,int dianacimiento,int mesnacimiento,int añonacimiento,String dni){//constructor clase Persona
    this.nombre=nombre;
    this.dianacimiento=dianacimiento;
    this.mesnacimiento=mesnacimiento;
    this.añonacimiento=añonacimiento;
    this.dni=dni;
}
//métodos de la clase persona
public String getNombre()
{
    return nombre;  //metodo get para obtener el nombre
}
public int getDianacimiento()
{
    return dianacimiento;   //metodo get para obtener el día de nacimiento
}
public int getMesnacimiento()
{
```

Esta clase, viene muy bien explicada con comentarios en el código. Resumidamente, crea personas con atributos como el nombre, DNI, fecha de nacimiento y probablemente añada el atributo provincia de residencia para completar la parte de la logística. Los métodos de esta clase son getters y setters de todos sus atributos.

Clase Productor:

```
/*********************************************************************************************
La clase Productores es una clase abstracta ya que no te puedes dar de alta en el sistema como un simple productor,
tienes que asociarte al tipo de productor, pequeño, grande y federado. Hay un atributo nuevo que es el de numerodetierras.
Este atributo es el que dependiendo de su valor va a definir las características de cada tipo de productor.
*********************************************************************************************/
public abstract class Productores extends Persona
{
 //campos de la clase Productores
 private int numerotierras;
 //constructor de la clase Productores
 public Productores(String nombre,int dianacimiento,int mesnacimiento,int añonacimiento,String dni,int numerotierras){
    super(nombre,dianacimiento,mesnacimiento,añonacimiento,dni);
    this.numerotierras=numerotierras;
}


}
```

La clase de Productor, la única novedad que tiene respecto a “Persona” es el atributo “numerotierras”. El número de tierras es la característica principal que diferencia entre un tipo de productor y otro. La clase de Productor es abstracta porque en el sistema se van a dar de alta directamente Productores asociados a un tipo de Productor (grande, pequeño, federado).

Clase Pequeños:

Primero vamos a comentar el campo de la clase y algunos atributos:

```
import java.util.ArrayList;

public class Pequeños extends Productores
{
    private Tierra[] tierras;
    private ListaProductos listaproductos;

    public Pequeños(String nombre,int dianacimiento,int mesnacimiento,int añonacimiento,String dni,int numerotierras){
        super(nombre,dianacimiento,mesnacimiento,añonacimiento,dni,numerotierras);
        Scanner scanner= new Scanner(System.in);
        tierras= new Tierra[numerotierras];
        listaproductos= new ListaProductos();

        //contadores de hectáreas totales y cantidad de productos distintos cultivados
        float hectáreasTotales=0;
        int productosDistintos=0;
        final int hectáreasPermitidas=5;
```

Importamos la interfaz de los ArrayList ya que los usaremos en esta clase. Primero declaro una matriz de solo columnas (realmente podría ser un ArrayList) de la clase Tierra.

```
public class Tierra
{
private float hectáreas;
private Producto producto;
private float rendimiento;

public Tierra(float hectáreas,Producto producto, float rendimiento){
    this.hectáreas=hectáreas;
    this.producto=producto;
    this.rendimiento=rendimiento;
}
public float getHectáreas() {
        return hectáreas;
    }
public Producto getProducto() {
        return producto;
    }
public float getRendimiento() {
        return rendimiento;
    }
}
```

Y luego creamos un objeto de la clase “ListaProductos” para almacenar en una lista todos los productos que tiene ese Productor. Una vez declarado, creamos el “Scanner” para recoger datos del usuario, y la matriz y arraylist con un tamaño o capacidad que depende del usuario. Por último, tenemos los contadores que son los que nos permitirán crear al Productor Pequeño respetando las características de estos (no más de 5 hectáreas y no más de 5 productos).

```
for (int i = 0; i < numerotierras; i++) {
    System.out.println("Introduce los datos de la tierra " + (i + 1));
    float hectáreas;
    float hectáreasDisponibles = hectáreasPermitidas - hectáreasTotales;
    do {
        System.out.print("Hectáreas en kilómetros (disponibles " + hectáreasDisponibles + "): ");
        hectáreas = scanner.nextFloat();
    } while (hectáreas > hectáreasDisponibles);
    hectáreasTotales += hectáreas;

    System.out.print("Nombre del producto: ");
    String nombreProducto = scanner.next().trim().toLowerCase();
    Producto producto;

    if (listaproductos.contieneProducto(new Producto(nombreProducto, false, 0))) {
        producto = listaproductos.getProducto(nombreProducto);
    } else {
        if(listaproductos.tamañoLista()>5){
        do{
            System.out.println("Se ha alcanzado el número máximo de productos diferentes, selecciona uno de los disponibles:");
            listaproductos.verProductos();
            String selectedProduct = scanner.next();
            producto = listaproductos.getProducto(selectedProduct);
        }while(!listaproductos.contieneProducto(producto));
    }
    boolean perecedero = generarTipoPerecedero();
    float precioKg = generarPrecioKg();
    producto = new Producto(nombreProducto, perecedero, precioKg);
    listaproductos.añadirProducto(producto);
    }

    System.out.print("Rendimiento por hectárea: ");
    float rendimiento = scanner.nextFloat();
    Tierra tierra = new Tierra(hectáreas, producto, rendimiento);
    tierras[i] = tierra;
```

Lo primero se repite este bucle según el número de tierras que tenga el Productor, donde cada Tierra tendrá sus características propias. Se van recogiendo los datos mediante Scanner y se van depositando en la matriz tierras. Se crea el objeto de Tierra y ese objeto se añade a la matriz tantas veces según el número de tierras. El Productor Pequeño tiene dos condicionantes, no puede tener más de 5 hectáreas y no puede tener plantados más de 5 productos diferentes. Por lo tanto, la variable “hectáreasDisponibles” es la variable que controla que no se sobrepasen las 5 hectáreas, lo podemos ver en el do-while. Al igual que en los if, si hay 1 producto que no coincide en la lista se añade y si la lista llega a más de 5 productos distintos tienes que poner uno de los que está en la lista existente.
Por último he creado métodos para una mejor interacción de las clases Productor que se da en Pequeños Grandes y Federados. Estas funciones son getTierras() que sirve para poder ver las Tierras que posee el productor. CrearTierras() que se repite el proceso que hay en el constructor al crear un productor; y eliminarTierras() si queremos quitar alguna tierra que pertenezca a algún productor en concreto. Lo que quiero es que todos los productos que se guardan en la lista de ese productor, se transfieran a una lista para todos los productores y que cuando los clientes quieran hacer un pedido (comprar un producto) tengan acceso a esa lista.

```
public void getTierras() {
    System.out.println("Tierras:");
    System.out.println("-------------------");
    System.out.println("| Número | Hectáreas | Producto | Rendimiento |");
    System.out.println("-------------------");

    for (int i = 0; i < tierras.length; i++) {
        Tierra tierra = tierras[i];
        System.out.printf("|   %2d   |   %.2f   |   %s   |   %.2f   |\n", (i + 1), tierra.getHectáreas(), tierra.getProducto(), tierra.getRendimiento());
    }

    System.out.println("-------------------");
}
 public void crearTierra(Scanner scanner) {
System.out.println("Introduce los datos de la nueva tierra:");
    float hectáreas;
    final int hectáreasPermitidas=5;
    float hectáreasTotales=0;
    float hectáreasDisponibles = hectáreasPermitidas - hectáreasTotales;
    do {
        System.out.print("Hectáreas en kilómetros (disponibles " + hectáreasDisponibles + "): ");
        hectáreas = scanner.nextFloat();
    } while (hectáreas > hectáreasDisponibles);
    hectáreasTotales+=hectáreas;
         System.out.print("Nombre del producto: ");
            String nombreProducto = scanner.next().trim().toLowerCase();
            Producto producto;
            if (listaproductos.contieneProducto(new Producto(nombreProducto, false, 0))) {
                producto = listaproductos.getProducto(nombreProducto);
            } else {
                boolean perecedero = generarTipoPerecedero();
                float precioKg = generarPrecioKg();
                producto = new Producto(nombreProducto, perecedero, precioKg);
                listaproductos.añadirProducto(producto);
```

```
public void eliminarTierra(Scanner scanner) {
    System.out.println("Seleccione el número de la tierra que desea eliminar:");
    getTierras();
    int numeroTierra = scanner.nextInt();

    // Verificar si el número de tierra es válido
    if (numeroTierra >= 1 && numeroTierra <= tierras.length) {
        // Eliminar la tierra del arreglo
        Tierra[] nuevasTierras = new Tierra[tierras.length - 1];
        int index = 0;
        for (int i = 0; i < tierras.length; i++) {
            if (i != (numeroTierra - 1)) {
                nuevasTierras[index] = tierras[i];
                index++;
            }
        }
        tierras = nuevasTierras;
        System.out.println("Tierra eliminada exitosamente.");
    } else {
        System.out.println("Número de tierra inválido. Inténtelo nuevamente.");
    }
}
```

Clase Grandes:

La clase Grandes es igual que “Pequeños” pero sin todas las restricciones de Productores Pequeños. Simplemente utilizamos Scanner para recoger datos de las tierras y creamos la matriz tierra de tamaño “numerotierras”. También están los métodos adicionales que tienen “Pequeños” y los productos se crean en una lista.

```
import java.util.Scanner;
import java.util.ArrayList;

public class Grandes extends Productores{
    private Tierra[] tierras;
    private ListaProductos listaproductos;

    public Grandes(String nombre,int dianacimiento,int mesnacimiento,int añonacimiento,String dni,int numerotierras){
        super(nombre,dianacimiento,mesnacimiento,añonacimiento,dni,numerotierras);
        Scanner scanner= new Scanner(System.in);
        tierras= new Tierra[numerotierras];
        listaproductos= new ListaProductos();

        for(int i=0; i<numerotierras; i++){
        System.out.println("Introduce los datos de la tierra " +(i+1));
        System.out.print("Hectáreas en kilómetros: ");
        float hectáreas = scanner.nextFloat();
        System.out.println("Producto: ");
        String producto = scanner.next();
        producto=producto.trim().toLowerCase();
        System.out.print("Rendimiento en porcentaje: ");
        float rendimiento = scanner.nextFloat();
        Tierra tierra = new Tierra(hectáreas,producto,rendimiento);
        tierras[i] = tierra;
    }
    }
```

Clase Federados:

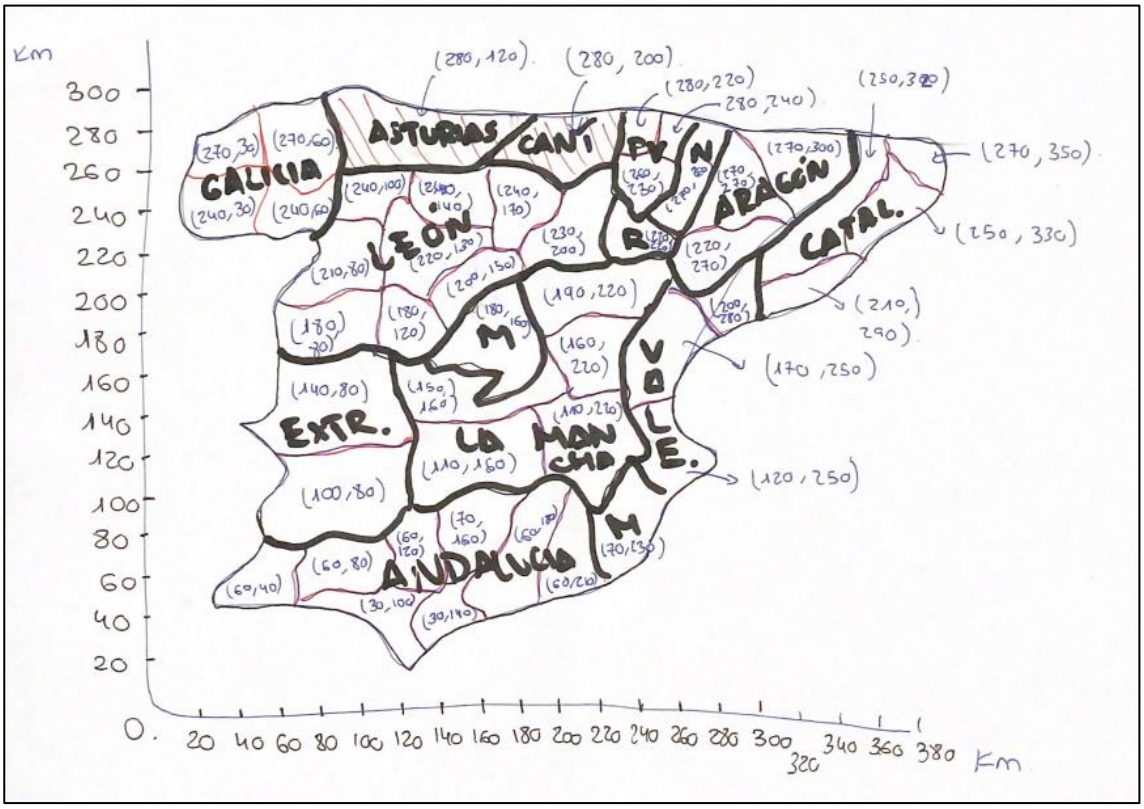
Se compone de dos personas que simplemente tienen una tierra con un producto y no pueden tener más. Aquí no he añadido un ArrayList de lista de productos porque solo puede haber uno. He añadido los métodos de mostrar la tierra, eliminarla o crear una nueva si han eliminado anteriormente.

```
import java.util.Scanner;

public class Federados extends Productores {
    private Tierra tierra;

    public Federados(String nombre1, int dianacimiento1, int mesnacimiento1, int añonacimiento1, String dni1,
            String nombre2, int dianacimiento2, int mesnacimiento2, int añonacimiento2, String dni2) {
        super(nombre1, dianacimiento1, mesnacimiento1, añonacimiento1, dni1, 1);
        Scanner scanner = new Scanner(System.in);
        float hectáreas;
        do {
            System.out.println("Introduce los datos de la tierra ");
            System.out.print("Hectáreas en kilómetros: ");
            hectáreas = scanner.nextFloat();
        } while (hectáreas <= 5);
        System.out.print("Nombre del producto: ");
        String nombreProducto = scanner.next().trim().toLowerCase();
        Producto producto;
        boolean perecedero = generarTipoPerecedero();
        float precioKg = generarPrecioKg();
        producto = new Producto(nombreProducto, perecedero, precioKg);
        System.out.print("Rendimiento por hectárea: ");
        float rendimiento = scanner.nextFloat();
        tierra = new Tierra(hectáreas, producto, rendimiento);
        System.out.println("Tierra creada exitosamente.");
    }
```

2.2. CLASE LOGÍSTICA

Esta clase tiene la complicación de cómo se define el trayecto que lleva la logística para entregar mercancías. He ideado una solución de la siguiente forma. He intentado plasmar el mapa peninsular de España, el mapa de provincias. Todo está en el paquete “MapaPeninsularEspaña”.

Para saber las distancias de provincias he hecho un mapa a mano alzada con distancias balanceadas para que tenga sentido la práctica y haya la misma probabilidad de trayectos de más de 100 km y de menos. Todas las personas y empresas que componen la cooperativa van a tener una provincia de localización.



Vamos a empezar a analizar las clases del paquete “MapaPeninsularEspaña”.

Clase Provincia:

Cada provincia tiene un nombre evidentemente, pero además, tiene una coordenada x e y para que luego, haciendo el módulo de un vector delimitado por dos puntos, sea la distancia entre provincias, es decir, la cantidad de kilómetros que tiene que hacer la logística.

```
package MapaPeninsularEspaña;
public class Provincia
{
    private String nombre;
    private float x;
    private float y;

    public Provincia(String nombre, float y, float x) {
        this.nombre = nombre;
        this.y = y;
        this.x = x;
    }

    public String getNombre() {
        return nombre;
    }

    public float getX() {
        return x;
    }

    public float getY() {
        return y;
    }
}
```

Clase ComunidadAutónoma:

```java
import java.util.ArrayList;

public class ComunidadAutonoma {
    private String nombre;
    private ArrayList<Provincia> provincias;

    public ComunidadAutonoma(String nombre, ArrayList<Provincia> provincias) {
        this.nombre = nombre;
        this.provincias = provincias;
    }
}
```

Esta clase tiene el atributo de nombre de la Comunidad Autónoma y un ArrayList de tipo Provincias.

Clase Mapa:

En la clase Mapa registramos 15 arraylist con los nombres de las comunidades autónomas y dentro, y sus provincias con sus coordenadas. Una vez están creadas todos estos arraylist, hacemos una última lista donde metemos todos los datos anteriores para que la información quede más comprimida.

```java
package MapaPeninsularEspana;
import java.util.ArrayList;
public class Mapa {
    public Mapa() {

        ArrayList<Provincia> andalucia = new ArrayList<>();
        andalucia.add(new Provincia("Almería", 60, 210));
        andalucia.add(new Provincia("Cádiz",30,100));
        andalucia.add(new Provincia("Córdoba",60,120));
        andalucia.add(new Provincia("Granada",60,180));
        andalucia.add(new Provincia("Huelva",60,40));
        andalucia.add(new Provincia("Jaén",70,160));
        andalucia.add(new Provincia("Málaga",30,140));
        andalucia.add(new Provincia("Sevilla",60,80));

        ArrayList<Provincia> aragon = new ArrayList<>();
        aragon.add(new Provincia("Huesca",270,300));
        aragon.add(new Provincia("Teruel",220,270));
        aragon.add(new Provincia("Zaragoza",270,270));

        ArrayList<Provincia> asturias = new ArrayList<>();
        asturias.add(new Provincia("Asturias",280,120));

        ArrayList<Provincia> cantabria = new ArrayList<>();
        cantabria.add(new Provincia("Cantabria",280,200));

        ArrayList<Provincia> castillaLaMancha = new ArrayList<>();
        castillaLaMancha.add(new Provincia("Albacete",110,220));
        castillaLaMancha.add(new Provincia("Ciudad Real",110,160));
```

Por último tenemos el método calcular la distancia que mediante “math” realizamos el módulo de un vector mediante la potencia al cuadrado y la raíz cuadrada. Calcular distancia es muy importante porque es el que nos diferencia cuando una logística es grande y cuándo es pequeña. Mediante los productos, que de forma aleatoria se dice si son perecederos, solucionamos toda la logística y sus tipos.

```
    ArrayList<ComunidadAutonoma> comunidadesAutonomas = new ArrayList<>();
    comunidadesAutonomas.add(new ComunidadAutonoma("Andalucía", andalucia));
    comunidadesAutonomas.add(new ComunidadAutonoma("Galicia", galicia));
    comunidadesAutonomas.add(new ComunidadAutonoma("Cantabria", cantabria));
    comunidadesAutonomas.add(new ComunidadAutonoma("Asturias", asturias));
    comunidadesAutonomas.add(new ComunidadAutonoma("Aragón", aragon));
    comunidadesAutonomas.add(new ComunidadAutonoma("La Rioja", larioja));
    comunidadesAutonomas.add(new ComunidadAutonoma("Navarra", navarra));
    comunidadesAutonomas.add(new ComunidadAutonoma("Murcia", murcia));
    comunidadesAutonomas.add(new ComunidadAutonoma("País Vasco", paisvasco));
    comunidadesAutonomas.add(new ComunidadAutonoma("Extremadura", extremadura));
    comunidadesAutonomas.add(new ComunidadAutonoma("Comunidad Valenciana", valencia));
    comunidadesAutonomas.add(new ComunidadAutonoma("Comunidad de Madrid", madrid));
    comunidadesAutonomas.add(new ComunidadAutonoma("Cataluña", cataluña));
    comunidadesAutonomas.add(new ComunidadAutonoma("Castilla la Mancha", castillaLaMancha));
    comunidadesAutonomas.add(new ComunidadAutonoma("Castilla y León", castillaLeon));

}

public double CalcularDistancia(Provincia provincia1, Provincia provincia2){

    double distancia= Math.sqrt(Math.pow(provincia2.getX() - provincia1.getX(), 2) +
                          Math.pow(provincia2.getY() - provincia1.getY(), 2));

    return distancia;

}
```

Además, he añadido funciones extra para una mejor implementación sobre la parte de la Logística y en el menú, para que los usuarios pueden seleccionar cuál es su ubicación y poder calcular distancias entre ellos.

```
    public void mostrarProvincias() {
        for (ComunidadAutonoma comunidad : comunidadesAutonomas) {
            System.out.println(comunidad.getNombre() + ":");
            for (Provincia provincia : comunidad.getProvincias()) {
                System.out.println("- " + provincia.getNombre());
            }
        }
    }

    public Provincia getProvinciaPorNombre(String nombre) {
        for (ComunidadAutonoma comunidad : comunidadesAutonomas) {
            for (Provincia provincia : comunidad.getProvincias()) {
                if (provincia.getNombre().equalsIgnoreCase(nombre)) {
                    return provincia;
                }
            }
        }
        return null; // Si no se encuentra la provincia
    }

    public ArrayList<Provincia> getProvincias() {
        ArrayList<Provincia> provincias = new ArrayList<>();
        for (ComunidadAutonoma comunidad : comunidadesAutonomas) {
            provincias.addAll(comunidad.getProvincias());
        }
        return provincias;
    }
```

Clase Logística: Tenemos 4 tipos de empresas logísticas. Perecederas pequeñas y grandes, No perecederas pequeñas y grandes. Por lo tanto, se definen por dos parámetros, la distancia que se resuelve mediante el tema del mapa y con la función CalcularDistancias(). Cuando se inicialice el menú se inicializa se inicializa un Mapa (El mapa de las provincias de España). Ahora con la clase Productos que será una clase ArrayList que almacena todos los productos, sin que se repitan, se decidirá mediante un Random si son perecederos o no. Mediante otra clase Random se decidirán los precios y por lo general, los perecederos serán más caros que los no perecederos.

La clase Logística es super simple, he interpretado que son empresas quiénes se encargan de la logística.

```
public class Logistica {
    private String nomEmpresa;
    private float saldo;

    public Logística(String nomEmpresa, float saldo) {
        this.nomEmpresa = nomEmpresa;
        this.saldo = saldo;
    }
    public String getNombre() {
        return nomEmpresa;
    }
    public float getSaldo() {
        return saldo;
    }

    public void setNombre(String nomEmpresa) {
        this.nomEmpresa = nomEmpresa;
    }

    public void recibirDinero(float cantidad) {
        saldo += cantidad;
    }
}
```

Las clases Perecederos y no Perecederos las he mantenido en el sistema pero no tienen nada especial y podía haber prescindido de ellas, pero era por mantener el esquema que se dio en la clase práctica.

Clase Pequeña:

Esta clase corresponde a la logística perecedera pequeña. Todas las logísticas van a estar programadas igual pero cada una tiene unos precios diferentes. En concreto sería algo así:

Logística Perecederos Grande > Logística Perecederos Pequeña > Logística No Perecederos Grande > Logística No Perecederos Pequeña.

Los precios se van a asignar con Random y los rangos de precios van a ser distintos según el tipo de Logística al que pertenezcan.

Esta clase es la primera en la que al usuario que se da de alta se le asigna una ubicación en forma de Provincia. Para ello utilizamos un mapa que es igual en todas las clases. Por lo tanto, lo nuevo de esta clase es la “instalación” del mapa y los métodos para que el usuario selecciones una de las provincias del mapa de forma adecuada.

```java
public class Pequeña extends Perecederos {
    private Provincia ubicación;
    private Mapa mapa;
    private String nombreEmpresa;

    public Pequeña(String nomEmpresa, float saldo, Mapa mapa) {
        super(nomEmpresa, saldo);
        this.nombreEmpresa=nomEmpresa;
        setMapa(mapa);
    }

    public void setMapa(Mapa mapa) {
        this.mapa = mapa;
    }

    public void seleccionarUbicacion() {
        // Mostrar las provincias disponibles para que el usuario seleccione
        System.out.println("Provincias disponibles:");
        mapa.mostrarProvincias();

        // Solicitar al usuario que ingrese el nombre de la provincia de ubicación
        String nombreProvincia = obtenerNombreProvincia();

        // Obtener la instancia de la provincia del mapa
        ubicación = mapa.getProvinciaPorNombre(nombreProvincia);

        if (ubicación == null) {
            System.out.println("Provincia no encontrada. Se establecerá una ubicación predeterminada.");
            ubicación = mapa.getProvincias().get(0); // Establecer una ubicación predeterminada
        }
    }

    private String obtenerNombreProvincia() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Ingrese el nombre de la provincia: ");
        String nombreProvincia = scanner.nextLine();
        return nombreProvincia;
```

Las clases Grande, NPequeña y NGrande son las de los demás tipos de logística que no hace falta comentar porque son iguales que Pequeña.

2.3. CLASE DISTRIBUIDOR Y CONSUMIDOR

Consumidor proviene de Cliente, que tiene las mismas características que Persona pero se le añade la variable saldo, variable que servirá como “cartera” para poder comprar productos (hacer pedidos).
La clase Distribuidor la he sacado de la herencia porque he interpretado que es una Empresa Distribuidora. Pero el funcionamiento de ambas, es el mismo. Tampoco voy a entrar en detalle porque lo de la ubicación funciona igual que en la anterior clase.

```java
public class Consumidores extends Cliente
{
    private Provincia ubicación;
    private Mapa mapa;
public Consumidores(String nombre,int dianacimiento,int mesnacimiento,int añonacimiento,String dni,float saldo,Mapa mapa){
    super(nombre,dianacimiento,mesnacimiento,añonacimiento,dni,saldo);
    setMapa(mapa);
}
public void setMapa(Mapa mapa) {
        this.mapa = mapa;
    }

public void seleccionarUbicacion() {
        // Mostrar las provincias disponibles para que el usuario seleccione
        System.out.println("Provincias disponibles:");
        mapa.mostrarProvincias();

        // Solicitar al usuario que ingrese el nombre de la provincia de ubicación
        String nombreProvincia = obtenerNombreProvincia();

        // Obtener la instancia de la provincia del mapa
        ubicación = mapa.getProvinciaPorNombre(nombreProvincia);

        if (ubicación == null) {
            System.out.println("Provincia no encontrada. Se establecerá una ubicación predeterminada.");
            ubicación = mapa.getProvincias().get(0); // Establecer una ubicación predeterminada
        }
            }
    private String obtenerNombreProvincia() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Ingrese el nombre de la provincia: ");
        String nombreProvincia = scanner.nextLine();
        return nombreProvincia;
    }
```

3. LA CLASE MENÚ Y LA CLASE MAIN

La clase main se va a basar en ejecutar la clase menú y empezaría con “public static void main”, por lo tanto, no hay mucho que comentar de la clase main. Esto se hace así por limpieza y por realizar un pequeño guiño a la programación orientada a objetos.

La clase menú, es más extenso pero es muy simple. Está inspirado en la asignatura del primer cuatrimestre de programación.
Va a haber menús para cada rol de la cooperativa agrícola: Productores, Logística y Consumidores.
Se utiliza while para que no termina la ejecución del menú o del programa hasta que no se pulse la tecla “Salir”.
Se utiliza do-while para que no se pase a un siguiente proceso hasta que el usuario no introduzca una opción válida.

```
import java.util.ArrayList;
import MapaPenínsularEspaña.Mapa;
import java.util.Scanner;
public class menu{
    private int opción;
    private ArrayList<Pequeños> productoresPequeños;
    private ArrayList<Grandes> productoresGrandes;
    private ArrayList<Federados> productoresFederados;
    private Pequeños pequeños;
    private Grandes grandes;
    private Federados federados;
    private Mapa mapa;
    private ListaProductos productosSistema;

    public menu(){
        Scanner scanner= new Scanner(System.in);
        productoresPequeños = new ArrayList<>();
        productosSistema= new ListaProductos();
        mapa=new Mapa();

        do{
        System.out.println("Bienvenido al sistema de administración de la Cooperativa Agrícola");
        System.out.println("Seleccione una opción del 1 al 5");
        System.out.println("1. Menú Productores");
        System.out.println("2. Menú Logística");
        System.out.println("3. Menú Distribuidores");
        System.out.println("4. Menú Consumidores");
        System.out.println("5. Salir");
        opción= scanner.nextInt();
        while (opción != 1 && opción != 2 && opción != 3 && opción != 4 && opción != 5)
                opción = scanner.nextInt();

            switch (opción) {
                case 1:
                    menuProductores(scanner);
                    break;
```

Se utiliza switch para todos los casos de procesos que se pueden dar en el menú.

```
do{
System.out.println("Bienvenido al sistema de administración de la Cooperativa Agrícola");
System.out.println("Seleccione una opción del 1 al 5");
System.out.println("1. Menú Productores");
System.out.println("2. Menú Logística");
System.out.println("3. Menú Distribuidores");
System.out.println("4. Menú Consumidores");
System.out.println("5. Salir");
opción= scanner.nextInt();
while (opción != 1 && opción != 2 && opción != 3 && opción != 4 && opción != 5)
        opción = scanner.nextInt();

    switch (opción) {
        case 1:
            menuProductores(scanner);
            break;
        case 2:
            // Menú Logística
            break;
        case 3:
            // Menú Distribuidores
            break;
        case 4:
            // Menú Consumidores
            break;
        case 5:
            // Salir del programa
            break;
    }
} while (opción != 5);
```

```
private void menuProductores(Scanner scanner) {
    do {
        System.out.println("Seleccione una opción del 1 al 7");
        System.out.println("1. Productor Pequeño");
        System.out.println("2. Crear Productor Pequeño");
        System.out.println("3. Productor Grande");
        System.out.println("4. Crear Productor Grande");
        System.out.println("5. Productor Federado");
        System.out.println("6. Crear Productor Federado");
        System.out.println("7. Salir");
        opción = scanner.nextInt();
        while (opción != 1 && opción != 2 && opción != 3 && opción != 4)
            opción = scanner.nextInt();

        switch (opción) {
            case 1: // Menú Productor Pequeño
                if (productoresPequeños.size() > 0) {
                    menuPequeños(scanner);
                } else {
                    System.out.println("No se ha creado ningún productor pequeño.");
                }
                break;
            case 2: // Crear Productor Pequeño
                crearProductorPequeño(scanner);
                break;
            case 3:
                menuGrandes(scanner);
                break;
            case 4: // Crear Productor Grande
                crearProductorGrande(scanner);
                break;
            case 5:
                menuFederados(scanner);
                break;
            case 6: // Crear Productor Federado
                crearProductorFederado(scanner);
```

```java
private void menuPequeños(Scanner scanner) {
    System.out.println("Seleccione el número de productor pequeño:");

    for (int i = 0; i < productoresPequeños.size(); i++) {
        System.out.println((i + 1) + ". " + productoresPequeños.get(i).getNombre());
    }

    System.out.println((productoresPequeños.size() + 1) + ". Volver al menú anterior");

    int opcionProductor = scanner.nextInt();

    while (opcionProductor < 1 || opcionProductor > productoresPequeños.size() + 1) {
        System.out.println("Opción inválida. Seleccione el número de productor pequeño o vuelva al menú anterior:");
        opcionProductor = scanner.nextInt();
    }

    if (opcionProductor == productoresPequeños.size() + 1) {
        return;
    }

    Pequeños productor = productoresPequeños.get(opcionProductor - 1);
    interactuarProductorPequeño(productor, scanner);
}
```

```java
private void interactuarProductorPequeño(Pequeños productor, Scanner scanner) {
    System.out.println("Nombre: " + productor.getNombre());
    System.out.println("DNI: " + productor.getDni());
    System.out.println("Fecha de nacimiento: " + productor.getDianacimiento() + "/" + productor.getMesnacimiento() + "/" + productor.getAñonacimiento());
    productor.getTierras();
    boolean salir = false;

while (!salir) {
    System.out.println("--------- Menú de Tierras ---------");
    System.out.println("1. Añadir tierra");
    System.out.println("2. Eliminar tierra");
    System.out.println("3. Salir");

    int opcion = scanner.nextInt();

    switch (opcion) {
        case 1:
            productor.crearTierra(scanner);
            break;
        case 2:
            productor.eliminarTierra(scanner);
            break;
        case 3:
            salir = true;
            break;
        default:
            System.out.println("Opción inválida. Seleccione una opción válida:");
            break;
    }
}
```

No voy a poner imágenes de todo el menú porque esto sería muy extenso, pero básicamente repito todo el rato lo mismo. Crear Eliminar y Ver.

Aquí un ejemplo de Crear:

```
private void crearProductorPequeño(Scanner scanner){
System.out.println("Ingrese el nombre del productor pequeño: ");
String nombre = scanner.nextLine();
System.out.println("Ingrese el día de nacimiento del productor pequeño: ");
int dianacimiento = scanner.nextInt();
scanner.nextLine();
System.out.println("Ingrese el mes de nacimiento del productor pequeño: ");
int mesnacimiento = scanner.nextInt();
scanner.nextLine();
System.out.println("Ingrese el año de nacimiento del productor pequeño: ");
int añonacimiento = scanner.nextInt();
scanner.nextLine();
System.out.println("Ingrese el DNI del productor pequeño: ");
String dni = scanner.nextLine();
System.out.println("Ingrese el número de tierras del productor pequeño: ");
int numerotierras = scanner.nextInt();
scanner.nextLine();
Pequeños nuevoProductor = new Pequeños(nombre,dianacimiento,mesnacimiento,añonacimiento,dni
productoresPequeños.add(nuevoProductor);
```

Aquí de Ver(usuarios registrados) y Eliminar(El que desee):

```
public void eliminarPequeñaPerecederos(Scanner scanner) {
    System.out.println("Empresas registradas:");
    for (Pequeña empresa : logísticaPerecederosPequeña) {
        System.out.println("- " + empresa.getNombreEmpresa());
    }
    System.out.println("Ingrese el nombre de la empresa a eliminar: ");
    String nombreEmpresa = scanner.nextLine();
    boolean empresaEncontrada = false;
    for (Pequeña empresa : logísticaPerecederosPequeña) {
        if (empresa != null && empresa.getNombreEmpresa().equals(nombreEmpresa)) {
            logísticaPerecederosPequeña.remove(empresa);
            System.out.println("La empresa " + nombreEmpresa + " ha sido eliminada.");
            empresaEncontrada = true;
            break;
        }
    }
    if (!empresaEncontrada) {
        System.out.println("No se encontró ninguna empresa con el nombre " + nombreEmpresa + "
    }
}
```

Lo último que me queda por hacer que está prácticamente hecho es la zona del menú de pedidos. Lo que pasa es que no tiene mucho orden y está hecho un poco desastre y me gustaría que quedase mejor.

```
            consumidorEncontrado = consumidores;
            break;
        }
    }
    if (nombreEncontrado) {
    Mostrar la lista de productos
        for (Producto producto : productosSistema) {
            System.out.println("- " + producto.getNombre());
        }

        String nombreProducto = scanner.nextLine();

        // Buscar el producto seleccionado
        Producto productoSeleccionado = null;
        for (Producto producto : productosSistema) {
            if (producto.getNombre().equalsIgnoreCase(nombreProducto)) {
                productoSeleccionado = producto;
                break;
            }
        }

        if (productoSeleccionado != null) {
            System.out.println("¿Cuánta cantidad en kg quiere del producto?");
            float cantidad = scanner.nextFloat();
            scanner.nextLine(); // Consumir el salto de línea

            // Realizar el pedido
            float costoTotal = productoSeleccionado.getPrecioKg() * cantidad;
            if (consumidores.getSaldo() < costoTotal) {
                System.out.println("El consumidor no tiene suficiente dinero para realizar el pedido");
            } else {
                consumidores.setSaldo(consumidores.getSaldo() - costoTotal);
                System.out.println("Pedido realizado exitosamente.");
            }
        } else {
            System.out.println("No se encontró ningún producto con el nombre " + nombreProducto + ".");
```

4. Otros Apuntes

Al entregar la práctica me he dado cuenta de que hay ciertas cosas que hubiese programado mejor o funciones que faltan. Muchas de estas funciones no tienen nada de especial, ni ningún conocimiento nuevo que aplicar, simplemente es repetir los procesos. Por lo tanto, quiero dejarlas aquí definidas para en un futuro si quiero mejorar la práctica realizarlo.

1. Pondría que las tierras de un productor tengan ubicación y que a la hora de hacer el pedido la distancia total se la suma de la distancia tierra-logística y logística-cliente para definir que tipos de logística utilizar.

2.En vez de repetir tanto método en las clases, crearía una clase auxiliar y para aplicar sus métodos en las otras clases crearía instancias y así se ahorraría mucho código.

3. No he hecho mucho caso al apartado de la práctica de crear tablas con historial de precios y diferencias de precios de productos y logística. Pero tampoco es muy complicado pues es recorrer ArrayList he imprimirlos en forma de tabla. Esto lo hago numerosas veces en la práctica. En el menú cuando se seleccionan productores, cuando se imprimen las tierras que pertenecen a los productores…

4. La clase Pedidos no la he usado porque en mi práctica la parte de pedidos no me ha quedado muy bien. Pero en esa clase la utilizaría para definir métodos

auxiliares. Cálculos de costes, imprimir pedidos, registrar pedidos, inventario de los clientes…

5. Base de Datos, he leído en el enunciado o en el foro que no está recomendado poner usuarios ya registrados. Que con el propio programa se registren los usuarios con las opciones, aunque en la clase práctica se recomendó el uso. No he realizado base de datos, pero en el menú en la zona de campos se podrían declarar fácilmente haciendo new Pequeño= (todos los datos del productor pequeño).

6. Resetear precios. Yo iba a crear una opción en la que mediante una clase se definan los precios de todos. Al seleccionar esa opción del menú, se crease una nueva instancia con nuevos precios otra vez asignados con Random y con límites mediante rangos. Al fin y al cabo, estas son las clases que estudian principalmente en el libro de POO y se pueden utilizar otras clases pero sería un poco trampa, a mi parecer, ya que no se aplicarían muchos otros conocimientos que intenta enfocar la asignatura.