Universidad Autónoma de Zacatecas

Trabajo Final: Clasificadores de objetos usando KNN

Estudiante:
José Francisco Hurtado Muro

Profesor:
Dr. Aldonso Becerra Sánchez

June 2, 2025

Tabla de Contenidos

Introducción

"La idea básica sobre la que se fundamenta el paradigma de clasificación KNN radica en que un nuevo caso se va a clasificar en la clase más frecuente a la que pertenecen sus K vecinos más cercanos. El paradigma se fundamenta por tanto en una idea muy simple e intuitiva, lo que unido a su fácil implementación, hace que sea un paradigma clasificatorio muy extendido."

1 Actividades que debe realizar el alumno:

1.1 Actividad inicial:

Lea cuidadosamente el trabajo completo antes de iniciar, ya que generará un **reporte IDC** que complemente el trabajo del código resultante solicitado.

1.2 Actividad 1:

Para este proyecto se piden los siguientes puntos:

1. El trabajo es individual.
2. Leer y entender el archivo base `Knn`.
3. Diseñar un esquema personalizado en donde se defina un escenario real para clasificar objetos utilizando el método **K-NN con pesado de variables**. El escenario debe incluir cientos de datos a clasificar en varias clases. Todo lo que se diseñe deberá:
   a) Ser desarrollado con Programación Orientada a Objetos (POO).
   b) Ser modular.
   c) Ser reutilizable.
   d) Utilizar Tipos Abstractos de Datos (TDA) y estructuras de datos creadas durante el curso, así como otras estructuras nuevas si son requeridas.
4. El sistema debe permitir introducir los datos de un nuevo objeto a clasificar y debe:
   a) Indicar a qué clase pertenece el objeto.
   b) Mostrar las clases disponibles.

   La interfaz puede ser en línea de comandos o gráfica (GUI).
5. **Fecha de entrega:** 2 de junio de 2025. La entrega debe incluir:
   a) Un video de máximo 10 minutos que muestre una explicación breve del código y del funcionamiento del programa. El video deberá subirse a Google Drive y se debe proporcionar el enlace (si la liga no funciona, automáticamente se califica con cero).
   b) Un reporte IDC donde se documente el trabajo realizado.
   c) El análisis y diseño del sistema que será codificado.
   d) El código completo del proyecto en un archivo .zip.

1.2.1 Lógica del algoritmo

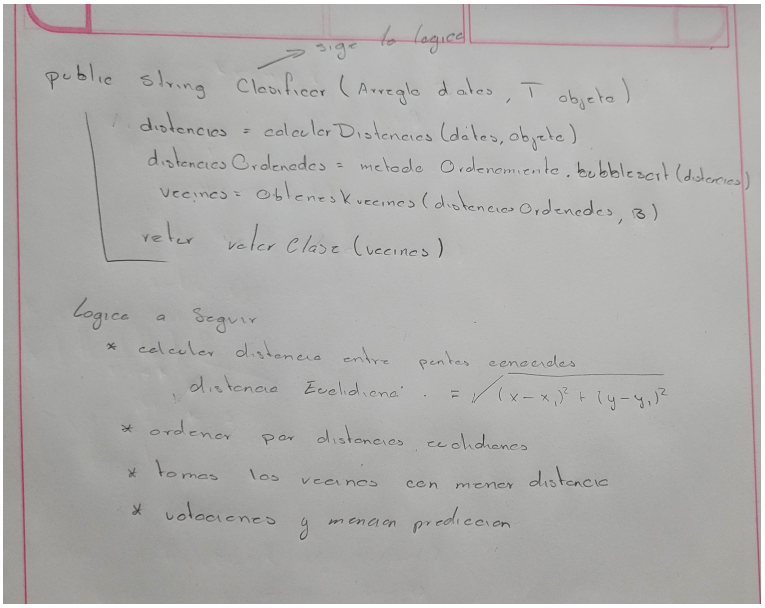

Figure 1: Lógica y Seudocodigo: Clasificar()

Explicación
Para este método lo que se hace es seguir la lógica de primero sacar las distancias de una ObjetoComparable desconocido a ObjetosComparables ya conocidos, esto lo hace por medio de la distancia Euclidiana y se va agregando a un objeto complejo llamado ObjetoDistacia que almacena un objeto cualquiera y la instancia, a continuación ordenamos los ObjtosDistancia en base a la distancia euclidiana y agarramos los mas pequeños, ya que estos significan que son los mas cercanos al ObjetoComparable a clasificar, se agarran los vecinos mas cercanos, **en este caso 3**, ya estando aquí lo que se hace es ver que clase o que objeto se repite mas y de esta manera se da una predicción de que datos puede ser

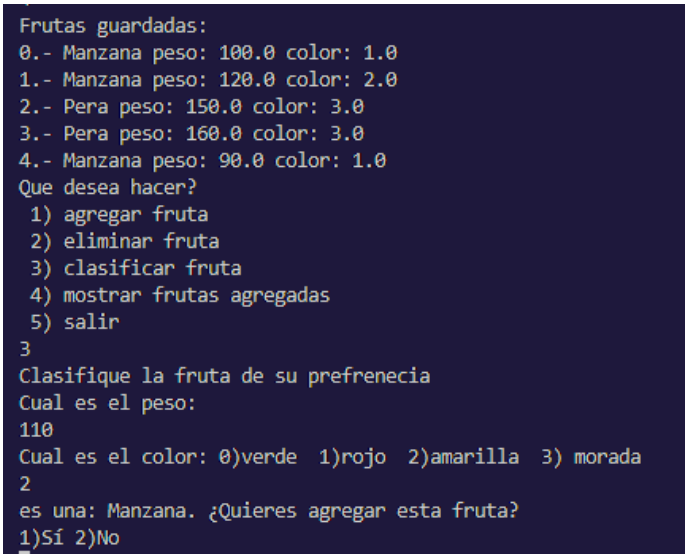

Figure 2: Funcionamiento: clasificar(Arreglo datos, T objeto)

1.2.2 Clases usadas para ser re-utilizable

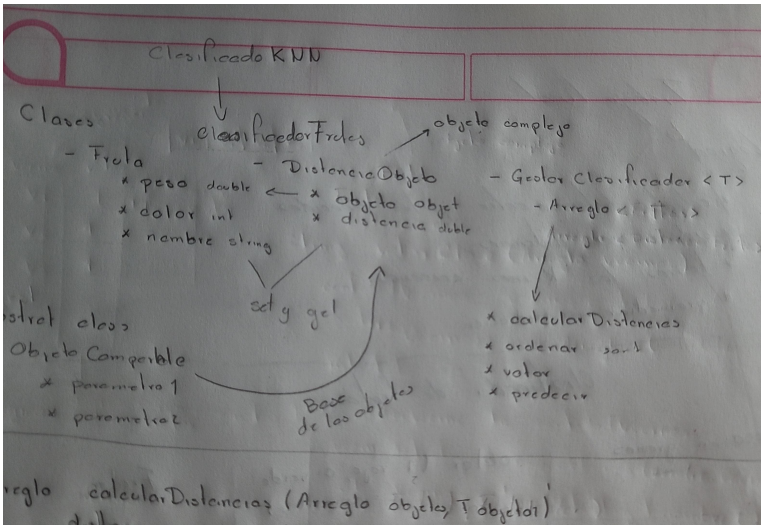

Figure 3: Clases

Explicación
Para poder hacer un código re-utilizable se creo un estándar en objetos comparables, **ObjetosComparables**, el cual contiene dos parametros doubles que nos ayudaran a poder ser usados en metodos como en **calcularDistancias**, estos parámetros son parametro1 y parametros2, que según el análisis que se haga ya se podrá escoger que sera cada cosa, para este caso parametro1, fue peso de una fruta y parametro2 fue el color representado por un numero, por lo cual podamos crear diferentes objetos y usarlos como un tipo de familia, lo cual nos sirve para un clase abstracta que tenemos que es **ClasificadorKNN** el cual solo puede ser usado con objetos de T que sean herencia de la clase **ObjtosComparables**, que en esta caso nos sirvió para crear el **ClasificadorFrutasKNN**, pero la logica fuerte pertenece a **ClasificadorKNN**, tal y como podemos ver en la figura 4

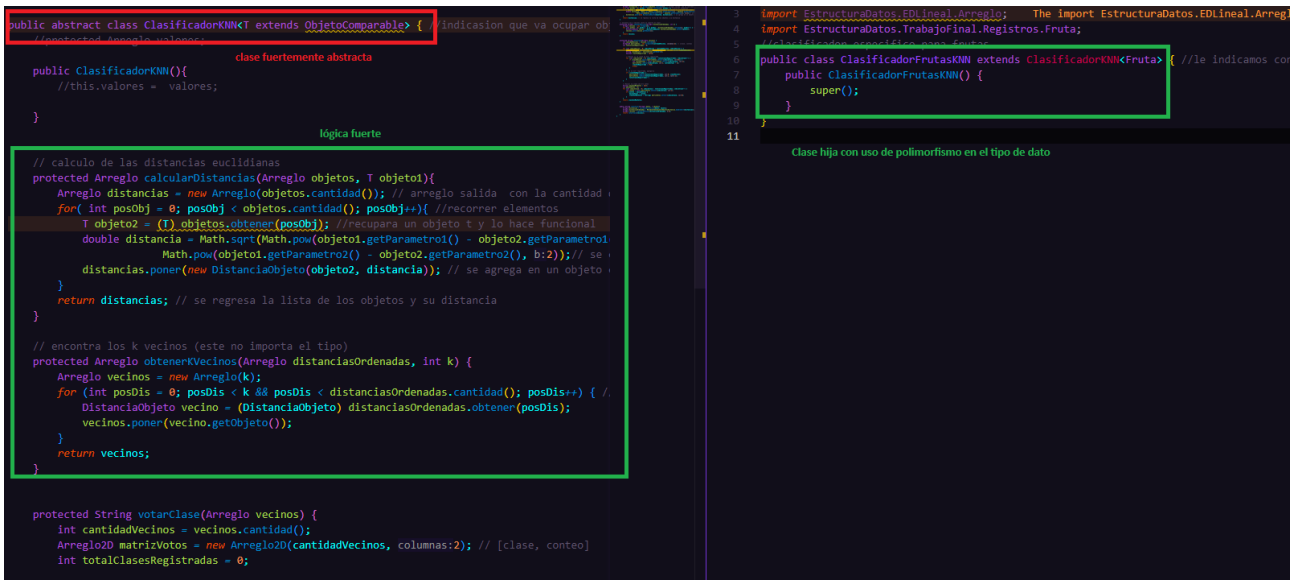

Figure 4: Muestra de como se ve el nivel de abstracción

1.2.3 calcularDistancias(Arreglo objetos, T objeto)

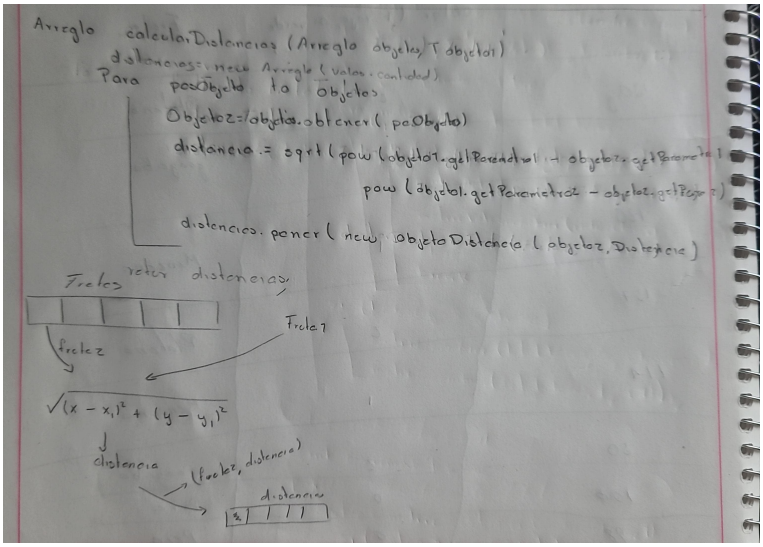

Figure 5: Analisis: calcularDistancias(Arreglo objetos, T objeto)

Explicación
En este caso lo que se hizo fue tomar el Arreglo mandado y del cual objeto por objeto iremos sacando sus dos parámetros ya hablados y previamente dicho que significaba cada uno de ellos, después de esto iremos aplicando la siguiente formula:

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

o en nuestro caso:

$$d = \sqrt{(peso_1 - peso_2)^2 + (color_1 - color_2)^2}$$

y al aplicarle esta formula a todos los elementos lo que hacíamos era ir metiendo los valores en DistanciaObjeto, los cuales se mandaban a un Arreglo que al finalizar de aplicarlos en todos, se retornaba como salida, tal como la siguiente figura 8.

Figure 6: Funcionamiento: calcularDistancias(Arreglo objetos, T objeto)

1.2.4 Ordenara los datos en base a su distancia

```
BUBBLESORT(A)
1    for i ← 1 to length[A] - 1
2        for j ← 1 to length[A] - i
3            if A[ j ] > A[ j+1]
4                exchange A[ j] <-> A[ j+1]
```

Figure 7: Ordenar datos: bubble sort, créditos al **Dr. Antonio García Domínguez**

Explicación
En este caso se buscó utilizar el método ya elaborado de *Tournament Sort*, sin embargo, presentó una limitante importante: no permite el uso de valores repetidos. Además, este método está diseñado para funcionar con objetos numéricos, por lo que no se pudo emplear directamente en nuestro contexto.

El uso de *Tournament Sort* habría sido ideal debido a su eficiencia temporal de orden $\mathcal{O}(n \log n)$. Como alternativa, se optó por utilizar el método *Bubble Sort*. Si bien este no tiene punto de comparación en cuanto a eficiencia con *Tournament Sort* —ya que su complejidad es de $\mathcal{O}(n^2)$—, se logró adaptar a las estructuras de datos utilizadas en el proyecto.

Con esta implementación, se consiguió ordenar los elementos de menor a mayor, gracias a que el método compara las distancias entre objetos y realiza los intercambios necesarios cuando corresponde.

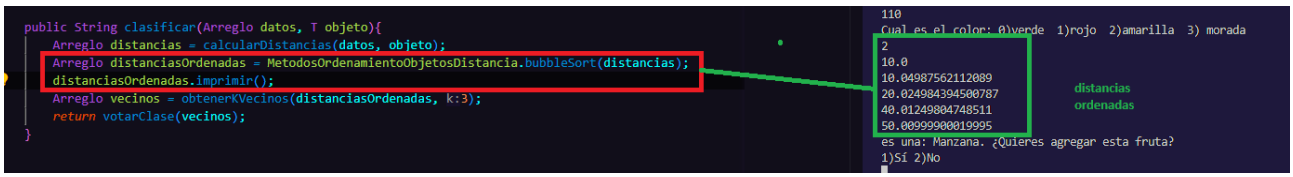

Figure 8: Funcionamiento: Ordenar datos

1.2.5 obtenerKVecinos(Arreglo distanciasOrdenadas, int k)

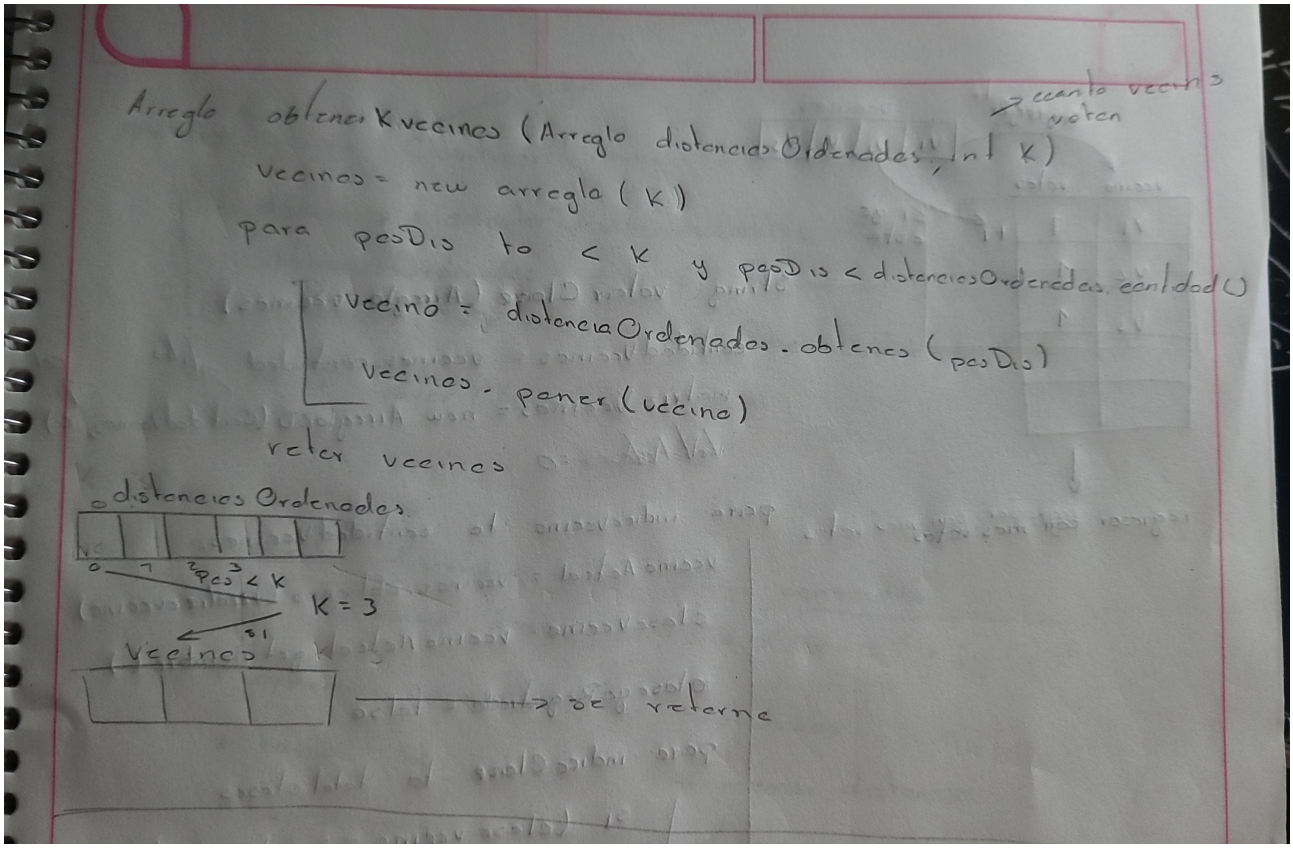

Figure 9: Análisis: obtenerKVecinos(Arreglo distanciasOrdenadas, int k)

Explicación

En este lo que se busco fue que este agarra los primeros vecinos en base de k, que para esta caso **K=3** asi que se se itero de 0 hasta k en los elementos de las distancias ordenadas sacando los 3 DistanciaObjetos y regresando en un Arreglo de tamaño k, tal y como la siguiente figura **??**

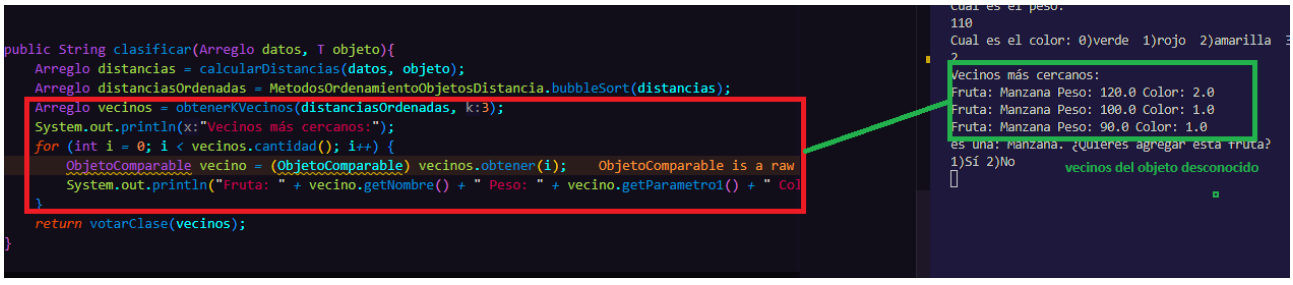

Figure 10: Funcionamiento: obtenerKVecinos(Arreglo distanciasOrdenadas, int k)

1.2.6 votarClase(Arreglo vecinos)

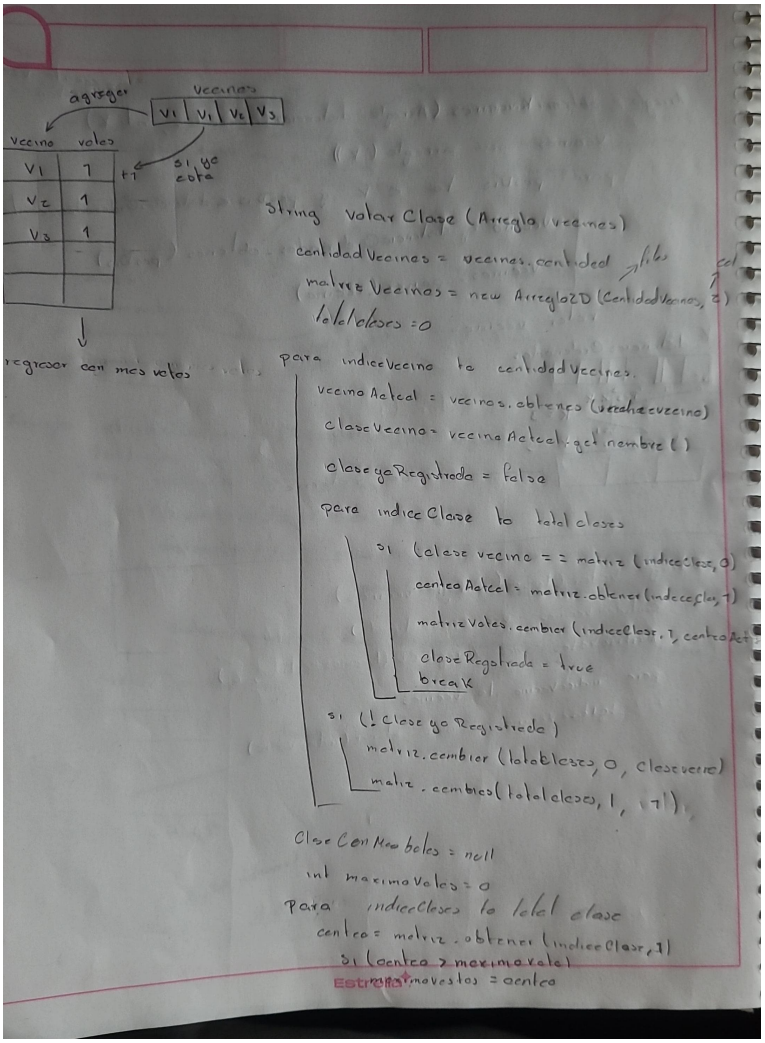

Figure 11: Análisis: votarClase(Arreglo vecinos)

Explicación

En este caso lo que se hace es crear un arreglo en 2 dimensiones, el cual en sus filas tendrá colo longitud la cantidad de elementos que haya de vecinos, que bien habíamos dicho serán 3, y 2 columnas, esto por que la segunda llevara el conteo de los elementos que vuelven a incidir, por lo cual recorremos el arreglo en búsqueda de las clases donde si la clase aun no ha sido agregada se agregara a la matriz, y si esta ya esta en la matriz lo que hara es que se la va a saltar pero le sumara 1 a las incidencias, al final se vera cual de loas clases es la que mas se repite y se regresa el nombre de ese objeto a modo de predicción de que el objeto a clasificar es de ese tipo, tal y como lo vemos en la figura 13

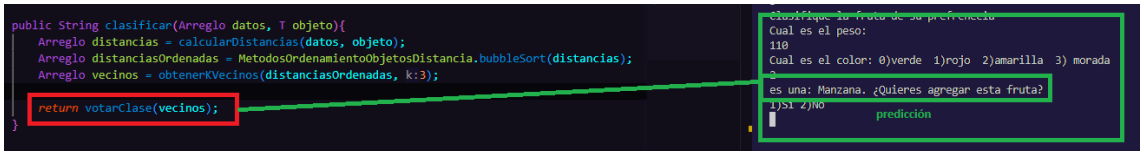

Figure 12: Funcionamiento: votarClase(Arreglo vecinos)

1.2.7 GestorClasificasionFrutas

La clase gestora unicamente tiene métodos para poder llevar a cabo de manera mas fácil y interactiva con el usuario, la funcionalidad dura es en los métodos anteriores, esto lo muestro ya que no tiene de mucho interés llamar funciones que ya están explicadas con la diferencia de que te muestra un mensaje de que vas a hacer y te sólita la entrada, por lo cual dejo a consideración de el revisar

```
public class GestorClasificasionFrutas {
    public Arreglo frutas;
    public ClasificadorKNN clasificador;    ClasificadorKNN is a raw type. References to generic type ClasificadorKNN<T> should be parameteri
    public GestorClasificasionFrutas(Arreglo frutas){
        this.frutas = new Arreglo(tamanio:70); //tamaño fijo
        this.frutas.agregarLista(frutas); // se agregan los valores de la lista nueva
        clasificador = new ClasificadorFrutasKNN();
    }

    public void agregarFruta(){
        Salida.salidaPorDefecto(cadena:"Cual es el peso:");
        double peso = Entrada.terminalDouble();
        Salida.salidaPorDefecto(cadena:"Cual es el color: 0)verde  1)rojo  2)amarilla  3) morada\n");
        double color = Entrada.terminalDouble();
        Salida.salidaPorDefecto(cadena:"Cual es el nombre de la fruta\n");
        String nombre = Entrada.terminalCadenas();
        Fruta nuevaFruta = new Fruta(peso, color, nombre);
        this.frutas.poner(nuevaFruta);
        Salida.salidaPorDefecto(cadena:"se agrego la fruta!\n");
    }

    public void mostraFrutas(){
        for (int elemento = 0; elemento<frutas.cantidad(); elemento++){
            Fruta fruta = (Fruta)frutas.obtener(elemento);
            Salida.salidaPorDefecto(elemento+".- " + fruta.getNombre()+" peso: "+fruta.getParametro1()+" color: "+fruta.getParametro2()+"\n")
        }
    }
    public void eliminarFruta(){
        Salida.salidaPorDefecto(cadena:"Ingrese la posicion de la fruta:\n");
        int posFruta = Entrada.terminalDouble().intValue();
        if (posFruta >= 0 && posFruta < this.frutas.cantidad()) {
            Fruta frutaEliminada = (Fruta)this.frutas.quitar(posFruta);
            Salida.salidaPorDefecto("Se elimino: " + frutaEliminada.getNombre()+ " en "+ posFruta +"\n");
        } else {
            Salida.salidaPorDefecto(cadena:"Posición inválida. No se eliminó ninguna fruta.\n");
        }
    }

    public void clasificarFruta() {
        boolean repetir = true;
        while (repetir) {
            Salida.salidaPorDefecto(cadena:"Cual es el peso:\n");
            double peso = Entrada.terminalDouble();
```

Figure 13: Funcionamiento: GestorClasificasionFrutas(fragmento)

2 Código Agregado - UML

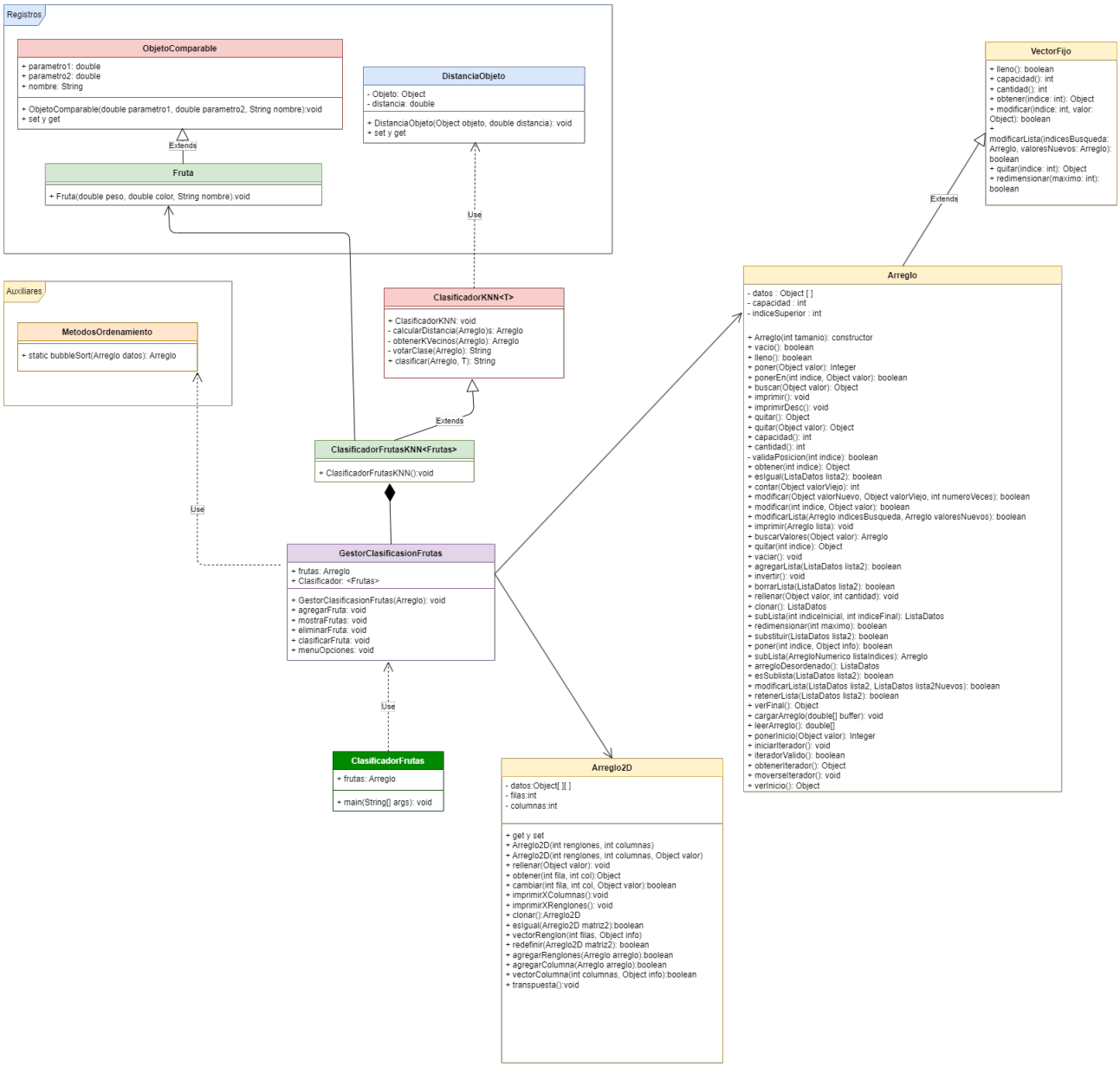

Figure 14: UML

3 Pre-evaluación del Alumno

Criterio	**Evaluación**
Cumple con la funcionalidad solicitada	**Sí**
Dispone de código auto-documentado	**Sí**
Dispone de código documentado a nivel de clase y método	**Sí**
Dispone de indentación correcta	**Sí**
Cumple la POO	**Sí**
Dispone de una forma fácil de utilizar el programa para el usuario	**Sí**
Dispone de un reporte con formato IDC	**Sí**
La información del reporte está libre de errores de ortografía	**Sí**
Se entregó en tiempo y forma la práctica	**si**
Incluye el código agregado en formato UML	**Sí**
Incluye las capturas de pantalla del programa funcionando	**Sí**
La práctica está totalmente realizada (especifique el porcentaje completado)	**100%**

Table 1: Evaluación de la práctica

4 Conclusión

Figure 15: **Si se puede imaginar se pude programar**

5 Referencias:

- https://youtu.be/FHHuo7xEeo4?si=q1k3LEiuuPwgPVDr