# Práctica 1: Programación orientada a objetos. Herencia e interfaces

## Introdución

En las prácticas de la asignatura **Programación Avanzada** vas a trabajar con los conceptos teóricos vistos en clase. Para ello te vas a introducir en el apasionante mundo del Aprendizaje Automático (*Machine Learning*) mediante la implementación de algunos algoritmos, bien conocidos, de esta disciplina.

## Sesiones

Esta práctica la vas a desarrollar en 2 sesiones.

## Objetivos

En esta primera práctica vas a trabajar con los conceptos presentados en clase de teoría sobre Programación Orientada a Objetos (POO), en particular con **Herencia** e **Interfaces**. Para ello vas a implementar un algoritmo bien conocido en Aprendizaje Automático: k-vecinos más próximos (KNN de sus siglas en inglés k-nearest neighbours).

Esta práctica se divide en dos partes, en las que implementarás:

1. Lectura de datos de un fichero con formato CSV.
2. Algoritmo de clasificación KNN.

## Lectura de datos desde ficheros CSV

### Introducción

En esta práctica vas a trabajar con ficheros en formato CSV (*Comma Separated Values*). El contenido de los ficheros CSV es texto que representa datos en formato tabular (filas y columnas). Los valores de cada columna están separados por comas.

En nuestro caso, cada fichero CSV contiene un conjunto de datos. Cada fila representa un ejemplar, también llamado instancia o muestra, y cada columna representa el valor de un atributo de ese ejemplar. El número de atributos puede ser distinto para ficheros CSV distintos, pero es el mismo para todos los ejemplares de un mismo fichero. En los ficheros que vamos a manejar, todos los ejemplares (filas) contienen valores válidos para todos los atributos (columnas).

En la Tabla American Express (https://www3.uji.es/~badenas/EI1017/practica1.html#american_express) tienes un ejemplo del contenido de un fichero CSV. Esta tabla tiene dos columnas: la primera columna representa el número de millas recorrido por la persona propietaria de una tarjeta American Express, la segunda columna es el gasto que la propietaria ha realizado con la tarjeta.

*Table 1. Fragmento de la tabla American Express*

| Miles | Dollars |
|---|---|
| 1211 | 1802 |
| 1345 | 2405 |
| 1422 | 2005 |

| Miles | Dollars |
|---|---|
| 1687 | 2511 |
| 1849 | 2332 |
| 2026 | 2305 |
| 2133 | 3016 |
| 2253 | 3385 |
| … | … |

Desde este miles_dollars.csv (https://www3.uji.es/~badenas/EI1017/datasets/miles_dollars.csv) te puedes descargar el fichero CSV con los datos.

La tabla Iris (https://www3.uji.es/~badenas/EI1017/practica1.html#iris) muestra un fragmento de otra tabla cuya última columna tiene la cabecera *class*. De nuevo, la primera fila de la tabla contiene los nombres de cada columna, y cada una de las siguientes filas contiene los datos de un ejemplar. Fíjate que en esta tabla, al contrario que en la anterior, la última columna tiene el nombre *class* que indica la clase a la que pertenece el ejemplar. Más concretamente, cada fila contiene la longitud y anchura del sepalo, y la longitud y anchura del pétalo de una flor, y la última columna indica la especie Iris de la flor a la que pertenece esta muestra (existen tres especies de Iris: setosa, versicolor y virginica). En este descripción Iris dataset (https://es.wikipedia.org/wiki/Conjunto_de_datos_flor_iris) tienes una descripción completa de los datos del fichero iris.csv (https://www3.uji.es/~badenas/EI1017/datasets/iris.csv)

*Table 2. Fragmento de la tabla Iris Dataset*

| sepal length | sepal width | petal length | petal width | class |
|---|---|---|---|---|
| 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| … | … | … | … | … |
| 7.0 | 3.2 | 4.7 | 1.4 | Iris-versicolor |
| 6.4 | 3.2 | 4.5 | 1.5 | Iris-versicolor |
| 6.9 | 3.1 | 4.9 | 1.5 | Iris-versicolor |
| 5.5 | 2.3 | 4.0 | 1.3 | Iris-versicolor |
| … | … | … | … | … |
| 6.3 | 3.3 | 6.0 | 2.5 | Iris-virginica |
| 5.8 | 2.7 | 5.1 | 1.9 | Iris-virginica |
| 7.1 | 3.0 | 5.9 | 2.1 | Iris-virginica |
| 6.3 | 2.9 | 5.6 | 1.8 | Iris-virginica |

| sepal length | sepal width | petal length | petal width | class |
|---|---|---|---|---|
| … | … | … | … | … |

*Conjuntos de datos etiquetados y no etiquetados.*

Vas a manejar ficheros donde cada ejemplar tiene asignada una etiqueta (nombre de la clase a la que pertenece el ejemplar), como en el caso de los datos mostrados en la tabla Iris (https://www3.uji.es/~badenas/EI1017/practica1.html#iris), al conjunto de datos que contienen estos ficheros se les llama *conjunto de datos etiquetados*. Por otro lado, también vas a manejar ficheros donde los ejemplares no poseen etiquetas, como en el caso de la tabla American Express (https://www3.uji.es/~badenas/EI1017/practica1.html#american_express), al conjunto de datos que contienen estos ficheros se les llama *conjuntos de datos no etiquetados*.

En nuestros caso, si el conjunto de datos está etiquetado, la columna con las etiquetas será siempre la última, con una etiqueta de cabecera en primera fila.

Resumiendo, para los ficheros CSV que vas a manejar:

1. Los ficheros CSV representan datos tabulados, con una primera fila de cabecera que contiene los nombres de los atributos.
2. Las siguientes filas contiene información sobre cada uno de los ejemplares, es decir, el valor de los atributos para ese ejemplar.
3. Los ejemplares pueden tener una última columna donde figura la etiqueta con el valor de la clase a la que pertenecen.

## Metodología

Vas a crear una clase *Table* para representar una tabla de datos. Cada una de las tablas tiene una cabecera y una lista que contiene los datos de la tabla. Cada elemento de la lista (ejemplar) estará representado por la clase *Row* y contendrá los datos de un ejemplar sin etiqueta. Para extraer y procesar la información del fichero CSV vas a utilizar una clase auxiliar a la que puedes llamar *CSV*. En la Figura 1 (https://www3.uji.es/~badenas/EI1017/practica1.html#uml_table) se muestra el diagrama UML de las clases.

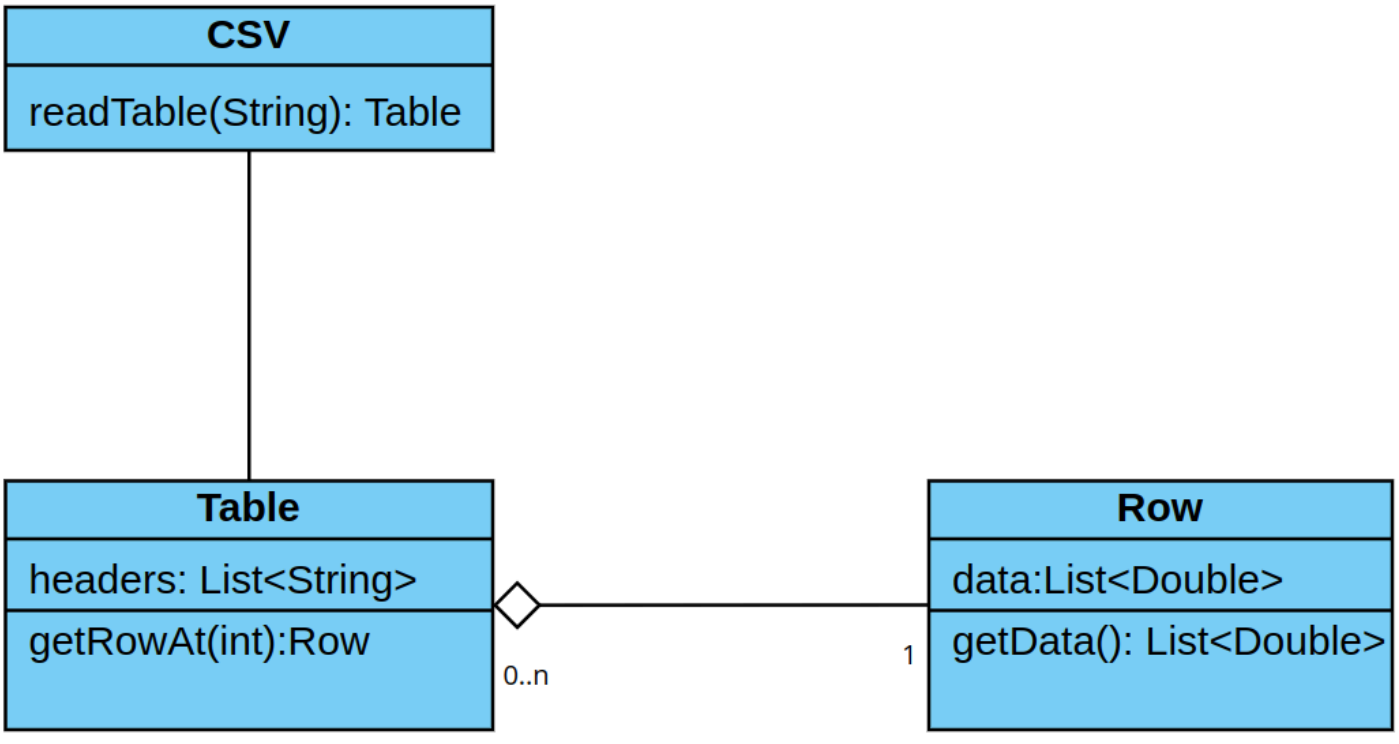


*Figura 1. UML de las clases Table, Row y CSV*

Debes definir, al menos, los métodos que aparecen en el diagrama. Además, por supuesto, puedes definir tantos otros métodos como te sea necesario. En particular el método:

JAVA

```
public Row getRowAt(int rowNumber)
```

te va a ser de utilidad para recuperar los datos de una fila de tu tabla de datos.

El siguiente paso es crear las clases necesarias para leer y almacenar los datos de un fichero CSV en el que los ejemplares tienen una etiqueta que indica la clase a la que pertenecen.

Para ello, lo primero que vas a hacer es extender la clase *Row* creando una nueva clase *RowWithLabel* que tenga un atributo para representar la etiqueta que tenga el ejemplar. En la Figura 2 (https://www3.uji.es/~badenas/EI1017/practica1.html#uml_table_extended) se muestra el diagrama UML de las clases.

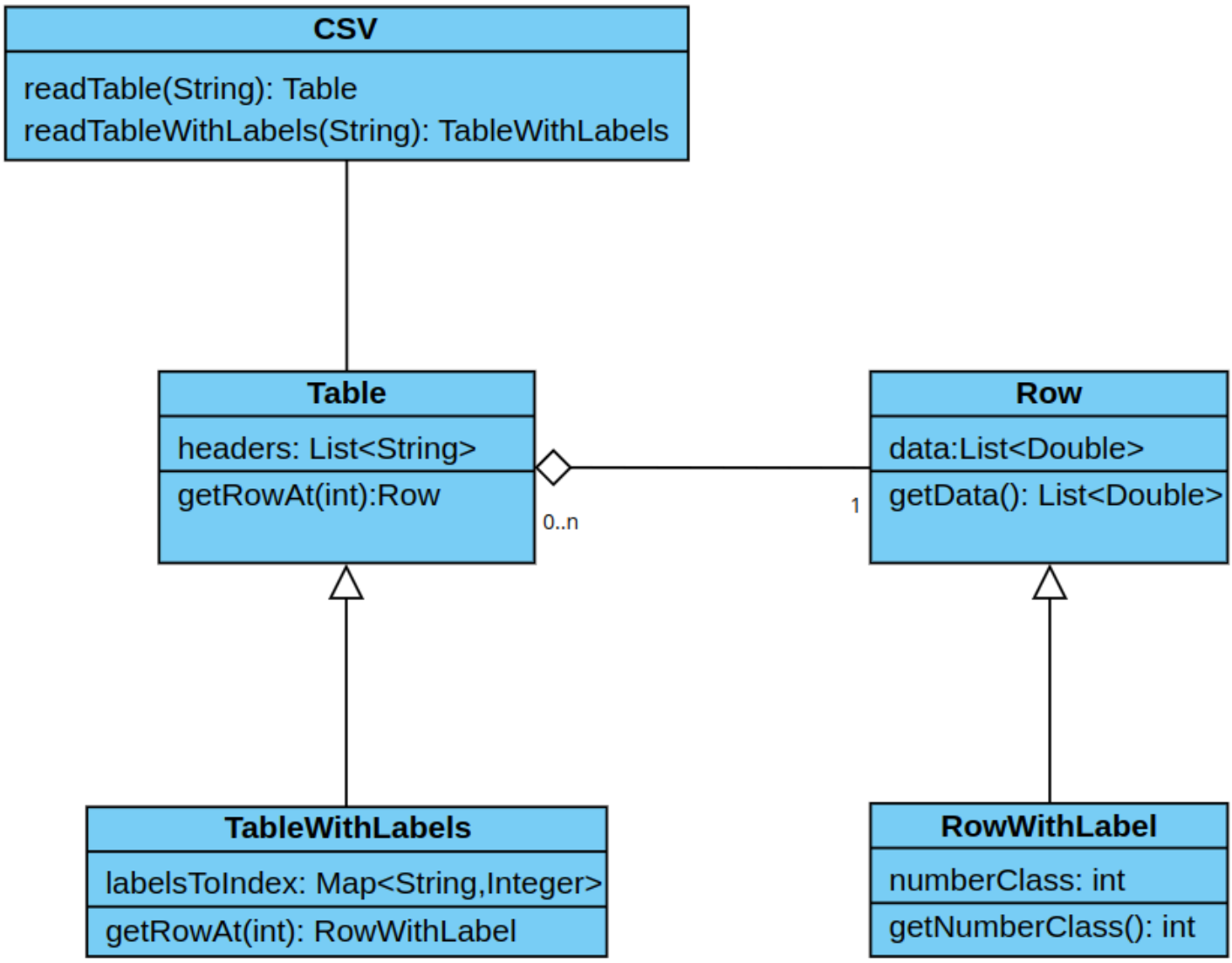


*Figura 2. UML de las clases TableWithLabels, RowWithLabels y CSV*

Si te fijas en las nuevas clases, observarás que además de heredar los atributos y métodos de sus clases base, incorporan otros miembros nuevos. *RowWithLabel* incorpora un atributo que representa el número de clase. Aunque las etiquetas son muy descriptivas, para manejar clases con los algoritmos resulta más conveniente utilizar números enteros para representarlas. Es por esto que para cada fila, además de guardar los valores de sus atributos, almacenaremos su número de clase en lugar de la etiqueta.

No obstante, no deseamos perder la información de las etiquetas. Por eso en la clase *TableWithLabels* recogeremos la correspondencia entre cada número de clase y su etiqueta. Esto lo haremos mediante el *Map* llamado *labelsToIndex*.

Los números de clase pueden generarse de manera automática cada vez que se añada una nueva fila. Esto puedes hacerlo a partir del número de elementos que contenga *labelsToIndex* en cada instante. Por ejemplo, la primera vez que añadas una fila, *labelsToIndex* tendrá tamaño cero. Puedes usar este valor como número de clase. Cuando añadas una segunda fila, el tamaño será uno, y usarás ese número como número de clase. Y así sucesivamente.

### Pruebas

La última parte es escribir las pruebas para verificar que tu código funciona correctamente.

Debes probar que:

1. El número de ejemplares (filas) leído es correcto.
2. El número de columnas es correcto.
3. El nombre de las etiquetas de las cabeceras es correcto.
4. El número que se asigna a cada fila es correcto.
5. Las filas que guardas en una tabla puedes recuperarlas conteniendo los mismos valores que guardaste.

Incluye algunas otras pruebas que a ti se te ocurran.

## Implementación del algoritmo KNN (K nearest neighbours)

### Introducción

El algoritmo KNN (https://es.wikipedia.org/wiki/K_vecinos_m%C3%A1s_pr%C3%B3ximos) es uno de los más sencillos, y más utilizados, en Aprendizaje Automático. Aquí te vamos a presentar una simplificación del mismo, la cual consiste en considerar que $k$ sea igual a $1$.

La idea básica del algoritmo es muy sencilla. Supón que has medido las características de 150 flores distintas de Iris y que conoces la especie de cada una de ellas, es decir, tus datos son los de la tabla Iris (https://www3.uji.es/~badenas/EI1017/practica1.html#iris). Ahora, supón que recoges una nueva flor, mides sus características y quieres saber cuál es la especie a la que pertenece ese ejemplar. Una manera sencilla de hacer una estimación de la especie a la que puede pertenecer el ejemplar, es comparar las características de tu nueva flor con todas las flores de tu tabla, y suponer (estimar) que tu nueva flor es de la misma especie que la de la flor a la que está más cercana, según sus características.

De una forma más algorítmica el procedimiento es:

1. Mide la distancia entre las características de tu nueva flor y todas las que tienes en la tabla.
2. Toma la flor de la tabla cuya distancia es menor a la de la nueva flor.
3. Asigna a la nueva flor la misma clase que la flor de distancia más cercana.

Así de sencillo.

Fíjate en que cuando lees los datos no tienes llevar a cabo ningún procesamiento. Es en el momento de estimar la especie a la que corresponde tu nueva flor cuando haces todo el cálculo.

### Metodología

En el algoritmo KNN vas a definir, al menos, los dos métodos siguientes:

```java
public void train(TableWithLabels data);
public Integer estimate(List<Double> data)
```

El primero de ellos tiene un argumento de la clase **TableWithLabels**. En realidad este método lo único que tiene que hacer es guardarse esta tabla para usarla después en la función **estimate**. De momento no te parecerá muy lógico el nombre de **train**. No te preocupes, porque en la siguiente práctica, cuando veamos algún otro algoritmo, lo entenderás.

En el segundo método (**Integer estimate(List<Double> sample)**) compararás el ejemplar que se pasa como argumento con todos los ejemplares cargados mediante el método **train**, y devolverás como estimación el número de clase del ejemplar más cercano en la tabla que guardaste con la función **train**.

La distancia la vas a calcular según la métrica euclidiana en $D$ dimensiones (una dimensión por cada característica). La distancia entre una nueva muestra $\vec{z}$ y un elemento en el fichero $\vec{x}$ se calcula como:

*Ecuación de la métrica Euclidea*

$$d(\vec{z},\vec{x}) = \sqrt{\sum_{i=1}^{D}(z_i - x_i)^2}$$

> *Métricas para calcular distancias*
> La euclidiana es una de las métricas que puedes utilizar para calcular la distancia entre dos puntos. Es la que utilizamos comúnmente para calcular la distancia entre dos puntos, y que aprendimos como teorema de Pitágoras. Pero existen muchas métricas distintas en el espacio euclidiano. Veremos algunas de ellas en la práctica dedicada a patrones de diseño.

## Pruebas

Debes realizar, al menos, las siguientes pruebas:

1. Que el cálculo de la distancia euclídea es correcto para distintos casos.
2. Que la función **estimate** asigna el ejemplar que le pasas a la clase correcta.

# Seguir aprendiendo

> En esta sección, lo que se propone son ejercicios que puedes realizar por tu cuenta si el tema de Machine Learning te está gustando.
>
> Ten en cuenta que lo que aquí se te propone no supone ninguna mejora de la nota, es, símplemente, alguna idea para que sigas aprendiendo sobre el tema.

## Más de un vecino

Hasta este momento con KNN, hemos basado nuestra estimación en un único ejemplar de de los de la base de datos que le pasamos con la funión **train**. Aunque hemos comparado con todos los ejemplares de la base de datos, hemos tomado como etiqueta para el nuevo ejemplar la misma que la del ejemplar más cercano en dicha base de datos. Tal y como te dijimos en la explicación del algoritmo, estábamos asumiendo la simplificación de $k = 1$. En el caso de que algún día necesites aplicar este método de Machine Learning, normalmente, conseguirás una mejor clasificación si empleas una mayor vecindad $k$.