# Práctica XII – Diagramas de clases

En esta práctica se analizarán, diseñarán y transformarán diagramas de clases a código.

Esta práctica se puede hacer hasta por grupos de 3.

## Ejercicio 1 (1,5 p cada ejercicio)

Realiza estos tres ejercicios usando una herramienta CASE (UMLet, Draw.io…)

1. Desde el instituto te piden que, como el mejor alumno de la historia de los ciclos de informática, realices un sistema de matriculación de los diversos alumnos. Antes de ponerte a programar en C#, como buen ingeniero de software que eres, vas a realizar un diagrama de clases. Las restricciones son:
   1. Se quieren guardar profesores, de los cuales guardaremos su nombre, apellidos, fecha de nacimiento, y asignaturas que imparte.
   2. De los alumnos guardaremos el nombre, los apellidos, el nacimiento y las notas.
   3. Se quiere almacenar un contador tanto de alumnos como de profesores.
   4. Los alumnos pueden ser de ciclo, o de fp básica. De los de ciclo queremos saber el grado al que pertenece y qué curso (primero, segundo) junto con su modalidad (presencial, semipresencial). De los de fp básica hay que almacenar el nombre de su madre y de su padre.
   5. Los alumnos pueden ser escolarizados en un curso, pudiendo tanto matricularse o desmatricularse del mismo. Los profesores también pueden matriculares o desmatricularse de un curso, lo que en este caso representa que pueden ser (o dejar de ser) tutores de un curso.
2. En un juego de ordenador existen 2 tipos de jugadores: los principiantes y los avanzados. Todos ellos deben tener un nombre y un número de vidas. Los principiantes se desplazan andando a unas coordenadas (x,y). Los jugadores avanzados además de andar también pueden conducir un vehículo para desplazarse más rápido a unas coordenadas. Cada vehículo tiene asociada una velocidad que puede ser leída y ajustada a un valor dado, pero no puede superar una velocidad máxima dada. La velocidad máxima sólo se podrá asignar una vez y no podrá ser modificada. Todos los atributos de las clases serán privados y tendrán métodos públicos para acceder a ellos (get/set) salvo que los requisitos indiquen lo contrario. Debe existir un método que se llame andar y otro conducir.
3. Desde el centro de astrofísica de Tenerife se te encomienda la tarea de generar un programa que gestione el envío y recepción de información de pequeños satélites que orbitan alrededor de las Islas Canarias:
   1. Se trabaja con la idea de cuerpo sólido, del cual queremos almacenar una imagen, la dirección, posición y velocidad.
   2. Un satélite, que es un tipo de cuerpo sólido, se puede controlar desde el observatorio, pudiendo aumentar, disminuir, girar hacia la izquierda, girar hacia la derecha y apagar el satélite.
   3. Para medir los diferentes valores (velocidad, dirección, imagen…) se usarán diferentes instrumentos, como una brújula, un velocímetro y un GPS. Cada uno de los instrumentos tratará la información de diferente forma.
   4. Los instrumentos se recogen en un panel de instrumentos, desde donde podemos activar, desactivar y visualizar la información obtenida.
   5. Para la obtención de la información se utiliza el patrón de arquitectura [*Observador*](https://refactoring.guru/design-patterns/observer), en el cual, podemos definir un mecanismo de suscripción de eventos para notificar a múltiples objetos (los subscriptores) sobre cualquier evento que esté ocurriendo sobre un objeto (llamado observador, o publicador). En este caso hay que modelar el concepto de ***Publicador*** y de ***Suscriptor***.
      1. Un publicador puede notificar a sus suscriptores, y permitir suscripciones o bajas. El publicador almacena una lista de los suscriptores.
      2. Un suscriptor tendrá el método de actualización, que será llamado por el publicador cuando el evento que se está observando ocurra.
      3. Un satélite es un publicador, pero podría ser que, si algún día se despliega un dron, o un cohete, también sean publicadores.
      4. Los suscriptores a los datos serán cualquiera de los instrumentos que tenemos en el sistema, pero teniendo en cuenta que cada instrumento procesará los datos de diferente forma

Consideraciones:

* Podéis implementar las clases en el mismo proyecto, pero en ficheros distintos.
* No hay que incluir el cuerpo de los métodos, sólo su cabecera.

## Ejercicio 2 (1,375 cada ejercicio)

Usando el plugin de diagramas de clase integrado en el Visual Studio, haz ingeniería inversa de los ejercicios de OOP hechos en la asignatura de programación (EJE0602) y en el examen.

Hay dos formas de hacer esto:

1. Si ya tenéis el código finalizado generar el diagrama a partir del mismo.
2. Si no habéis hecho algún ejercicio (probable) hacer el diagrama y después pasarlo a código.
3. Si tenéis el código a medias, o mal, o sin finalizar, pasar lo que tengáis a diagrama y terminarlo de forma gráfica.

Consideraciones:

1. Si hacéis el ejercicio por grupos, seleccionad el proyecto del integrante que prefiráis.
2. Hay que entregar tanto el código finalizado como el diagrama, no sirve con entregar una de las dos cosas porque habéis copiado código de un lado hacia otro.

## Fecha de entrega

Hay que entregar un único proyecto de VS con los apartados de cada ejercicio hecho, junto a los. uxf de los diagramas de clase del primer ejercicio.

Como hay muchos apartados, valoraré con penalización de 1 punto que las clases sigan la nomenclatura (***EjercicioX\_ApartadoY.cs***), siendo X e Y el número asociado a cada ejercicio y apartado.

Hay que entregar un enlace al repositorio de GitHub con toda esta información (proyecto + ficheros .uxf)

La fecha límite de entrega será el 04/02/2023 a las 23:59 como mucho.