UTN
Facultad Regional Córdoba

TUP

Tecnicatura Universitaria en Programación

# PROGRAMACIÓN I

Unidad Temática N°3:
Herencia de Clases

Guía de Estudio
1° Año – 1° Cuatrimestre

# Índice

## HERENCIA DE CLASES CON C#

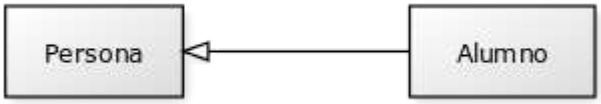
Basado en el Paradigma Orientado a Objetos, diseñe una solución escrita en lenguaje de programación C# para cada uno de los problemas que se plantean a continuación.

En cada caso defina:

- Clases que incluyan atributos, propiedades, constructor sin parámetros, constructor con parámetros, métodos de control y método ToString().
- Relaciones que implementen el modelo planteado en cada caso.
- Desarrollar una Interfaz Gráfica de Usuario (mediante WinForms) que permita implementar el modelo de objetos diseñado. Tener en cuenta que al momento de crear los objetos se deberán hacer las validaciones de datos correspondientes y utilizar los componentes visuales adecuados para cada caso.

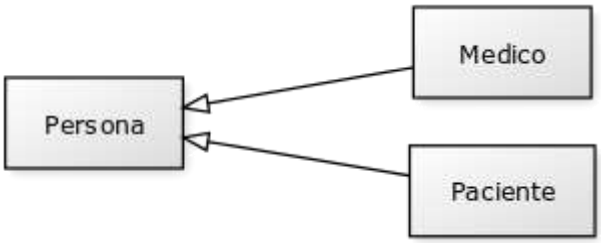
### Problema 3.1:

**Alumnos**. Dados el nombre, legajo, DNI, teléfono y tres notas de alumnos, calcular su promedio y mostrar todos sus datos.




### Problema 3.2:

**Clínica**. Desarrollar un programa que permita registrar el nombre, DNI, teléfono y sexo de médicos y pacientes de una clínica. Para médicos, cargar también su especialidad (1-Traumatología, 2-Pediatría, 3-Cardiología) y costo de la consulta y para pacientes la obra social (1-Apross, 2-Met, 3-Osde, 4-Otras) y datos de su historia clínica.




Además, indicar la cantidad de médicos y pacientes registrados, el costo promedio de las consultas médicas y el porcentaje de pacientes que tienen Osde como obra Social.

### Problema 3.3:

**Banco**. Dados el nombre, DNI, sexo, código y límite de crédito de un cliente y el número, tipo (1-caja de ahorro, 2-cuenta corriente) y saldo de una cuenta, desarrollar un programa que permita registrar las cuentas de los clientes de un banco de acuerdo al siguiente diagrama.


Además, calcular la cantidad de cajas de ahorro y de cuentas corriente, saldo promedio para caja de ahorro, para cuenta corriente y general, porcentaje de clientes femeninos y el cliente con mayor límite de crédito.

### Problema 3.4:

**Inmobiliaria.** Dados los datos de un propietario: tipo y número de DNI, nombre y sexo; y de un inmueble: metros, costo por metro y tipo de inmueble (1-Casa, 2-Dpto, 3-Lote), desarrollar un programa que permita registrar los propietarios con sus inmuebles de acuerdo al siguiente diagrama.


Además, determinar la cantidad de casas, departamentos y lotes, su valuación promedio, el valor promedio de todos los inmuebles, el porcentaje de casas, la cantidad de mujeres con departamento, el propietario con inmueble más valioso y el propietario con lote más pequeño.

NOTA: tener en cuenta que la valuación de los inmuebles resulta de la cantidad de metros por el costo por metro en cada caso.

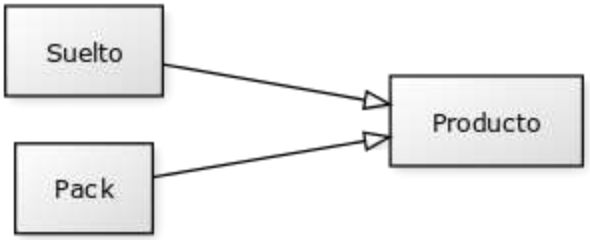
### Problema 3.5:

**Veterinaria.** Dados los datos del dueño (nombre, sexo, código) y de su mascota (edad, nombre y tipo: 1-perro, 2-gato, 3-araña, 4-iguana), desarrollar un programa que permita registrar los dueños con sus mascotas de acuerdo al siguiente diagrama.


Además, determinar la cantidad, porcentaje y edad promedio de cada tipo de mascota, la cantidad de mujeres con iguanas, la cantidad de hombres con gatos, el dueño con perro más viejo y la mascota más joven.

### Problema 3.6

**Productos.** Dados el código, marca, precio unitario y tipo (suelto o pack) de productos, calcular la cantidad de productos sueltos, la cantidad de productos en packs, el precio promedio de productos sueltos, el porcentaje de packs y el código y marca del pack más costoso. Los productos sueltos tienen medida y los packs cantidad. Implementar las clases Producto, Pack y Suelto y el método abstracto getPrecio(). En cada caso, el método devuelve (cantidad * precio) para packs y (medida * precio) para productos sueltos. Con ello crear un arreglo de productos. Mostrar dicho arreglo en una lista y mostrar el precio total.




## EJERCICIOS INTEGRADORES

### Problema 3.7

**Biblioteca.** Crear una clase Libro que contenga los siguientes atributos:

- ISBN
- Titulo
- Autor
- Número de páginas

Crear sus respectivos métodos get y set correspondientes para cada atributo. Crear el método toString() para mostrar la información relativa al libro con el siguiente formato:

“El libro con ISBN creado por el autor tiene páginas”

Crear una clase Adicional Biblioteca que permite modelar nuestro ordenamiento de libros. Agregar los siguientes comportamientos:

- Un constructor con un parámetro, que permite definir el tamaño de la estantería (cantidad de libros). Validar que la cantidad sea mayor que cero.
- Un método agregarLibro(unLibro) que permita agregar un libro a la biblioteca siempre que la estantería no se encuentre llena.
- Un método mostrarListado() que permita retornar el estado de la biblioteca como una cadena: [Libro1 |Libro2 |Libro3… |]

## Problema 3.8

**Biblioteca Parte II.** Supongamos ahora que en nuestra biblioteca también almacenamos Revistas. De cada revista se conoce: código, título y número de páginas.

- Crear el método toString() para mostrar la información relativa a la revista con el siguiente formato: “La revista título tiene páginas”

Refactorizar la clase Biblioteca para que podamos gestionar tanto libros como revistas. Definir adicionalmente los comportamientos:

- buscarPublicacion(unTitulo) que dado un título permita buscar una publicación en nuestra biblioteca. Si existe mostrar todos sus datos. Caso contrario informar con un mensaje.
- contarLibros(nPaginas) que permita conocer la cantidad de libros de la biblioteca con más n páginas, siendo n un parámetro ingresado como parámetro

## Problema 3.9

**Transporte de cargas**. Una empresa de transporte de cargas necesita un software que la ayude a organizarse con la carga de los camiones que maneja. La empresa puede recibir packings y bidones que transportan líquido.

- Un packing es una estructura de madera que arriba tiene un montón de cajas, se envuelve todo con plástico para que no se desbanden las cajas. Todas las cajas tienen el mismo peso. El peso de un packing es (peso de cada caja * cantidad de cajas) + peso de la estructura de madera que va abajo. Para cada packing se informa que llevan las cajas, p.ej. Material de construcción.
- El peso de un bidón es su capacidad en litros por la densidad (o sea, cuántos kg pesa un litro) del líquido que se le carga. Los bidones van siempre llenos hasta el tope.
- Cada camión puede llevar hasta una carga máxima medida en kg. Además, cada camión puede: estar disponible para la carga (en cuyo caso ya puede tener cosas cargadas), estar en reparación, o estar de viaje.

**Algunos requerimientos:**

- Subir una carga al camión, donde la carga puede ser un packing, una caja suelta, o un bidón. Considerar que no se puede saturar un camión con más peso de lo que su carga máxima permite.

- Bajar una carga del camión, siempre que el camión se encuentre disponible con cargas y que a su vez la carga se encuentre presente dentro de él.
- Conocer el total de cargas de un camión en todo momento y su peso.
- Saber si un camión está listo para partir, que es: si está disponible para la carga, y el peso total de lo que tiene cargado es de al menos 75% de su carga máxima.