UNIVERSIDAD DE MURCIA

# Tecnología de la Programación

## Ejercicios adicionales

## Tema 3

*Autores: Alejandro Buitrago López (alejandro.buitragol@um.es), Sergio López Bernal (slopez@um.es), Javier Pastor Galindo (javierpg@um.es), Jesús Damián Jiménez Re (jesusjr@um.es) y Gregorio Martínez Pérez (gregorio@um.es)*

Dpto. de Ingeniería de la Información y las Comunicaciones

Curso 2025-2026

Última modificación:
26 de noviembre de 2025

# Tema 3: Ejercicios adicionales

**Índice**



Estos ejercicios adicionales están diseñados para trabajar de manera complementaria y profunda los conceptos trabajados durante el **Tema 3**, como los distintos **tipos de relaciones**, **delegación**, **clonación de objetos**, la **modularidad** y **herencia**.

## 3.1 Red social

En este ejercicio desarrollaremos una versión simplificada de una red social. Define tu propia estructura de proyecto usando módulos y paquetes.

En primer lugar, crea una clase en Python que represente un **Usuario**. Un **Usuario** se caracteriza por su nombre, una lista de usuarios seguidos, otra lista de seguidores, una contraseña (que no se podrá cambiar), un token único para cada usuario generado de manera aleatoria cuando el usuario se registra y el correo del usuario. Si no se proporciona, deberá ser: `nombre_del_usuario@correo.com`.

**Pregunta:** ¿Qué tipo de relación existe entre el **Usuario** y los otros usuarios que sigue o que lo siguen?

De igual forma, tendremos dos tipos de **Usuario**:

- **Usuario regular**: Representa a un usuario estándar que puede seguir y ser seguido, interactuar con posts y comentarios, pero no tiene privilegios adicionales.
- **Usuario administrador**: Representa un administrador. Además de las funcionalidades estándar, un administrador podrá crear y gestionar comunidades. El administrador podrá realizar acciones que los usuarios regulares no pueden, como eliminar o editar posts en las comunidades que administran.

**Pregunta:** ¿Cuál es la relación entre **Usuario**, **Usuario regular** y **Usuario administrador**?

Dentro de nuestra red social, tendremos **comentarios**. Un comentario se caracteriza por:

- El **usuario** que ha creado el comentario.
- El **contenido** del comentario en formato textual.
- La **fecha de creación**.
- Una lista de **usuarios que han dado "me gusta"** al comentario.

Posteriormente, vamos a añadir la funcionalidad de **posts** o publicaciones a nuestra red social. Para ello, crea una clase llamada **Post**. Un **Post** se caracteriza por:

- El **usuario** que ha creado el post.
- El **contenido** del post en formato textual.
- La **fecha de creación**.
- Una lista de **usuarios que han dado "me gusta"** al post.
- Los **comentarios asociados** al post.

**Pregunta:** ¿Qué ocurre con los comentarios si se elimina el post? ¿Tienen sentido fuera del contexto del post? ¿Es una relación de asociación, agregación o composición? ¿Por qué?

También introduciremos el concepto de **comunidad**, esto es, un grupo de usuarios con intereses similares que comparten contenido. Por lo tanto, una comunidad se caracterizará por:

- Su **nombre**.

- Una **breve descripción**.
- Los **Usuarios** que pertenecen a la comunidad.
- El **administrador** del grupo, que es un **Usuario**.
- Los **Post** realizados dentro de la comunidad.

**Pregunta:** Si el administrador de la comunidad es un **Usuario**, ¿Qué tipo de relación existe entre **Comunidad** y **Usuario**? ¿Es posible que el administrador cambie sin que la comunidad desaparezca? ¿Qué implicaría eso en términos de agregación o asociación?

Por último, implementaremos un **sistema de notificaciones** dentro de la red social que hemos desarrollado. Cada vez que un usuario publica un post o realiza un comentario, se generará una **notificación** que informa a sus seguidores de la nueva actividad. Una **notificación** se caracteriza por:

- Su **mensaje** (ej: Alex ha publicado un nuevo post).
- La **fecha en la que se envía**.

Implementa las siguientes funcionalidades:

1. **Seguir usuarios**

   Crea un nuevo método llamado `seguir_usuario`, que implementará la funcionalidad típica de las redes sociales de seguimiento de usuarios, similar a redes como X, Instagram, etc. Este método debe permitir que un usuario (identificado como *usuario que sigue*) siga a otro (identificado como *usuario seguido*).

   Para ello, el método deberá:

   - Agregar al *usuario que sigue* a la lista de seguidores del *usuario seguido*.
   - Agregar al *usuario seguido* a la lista de seguidos del *usuario que sigue*.

   Antes de realizar estas acciones, se debe verificar que:

   - El *usuario que sigue* no esté ya en la lista de seguidores del *usuario seguido*.
   - El *usuario seguido* no esté ya en la lista de seguidos del *usuario que sigue*.

   Si ambas condiciones se cumplen, se realiza la operación de seguimiento. En caso contrario, se debe indicar que la acción no es válida. Esto garantiza la integridad de los datos.
2. **Crear posts, comentarios y reacciones**

   Crea un método para permitir que los usuarios creen **posts** y **comentarios** en la red social, y también para que puedan dar *me gusta* tanto a un **post** como a un **comentario**.

   **Pregunta:** ¿En qué clases deben estar estos métodos?
3. **Notificar**

   Crea un método para enviar **notificaciones**. Este método será llamado cada vez que un usuario cree un **post** o comente en uno, y enviará la notificación a todos los seguidores del usuario.

   **Pregunta:** ¿Qué tipo de relación supone la creación de este método?
4. **Recomendación de usuarios**

   A continuación, implementaremos la funcionalidad de recomendación de usuarios para seguir en una red social, basada en los amigos en común entre los usuarios. El sistema sugerirá que un usuario siga a aquellos que siguen a personas que el usuario ya está siguiendo.

   La recomendación debe basarse en las conexiones indirectas: si un usuario está siguiendo a alguien que sigue a otra persona, esa persona debe ser recomendada para seguir. Asegúrate de que el usuario no reciba recomendaciones para seguir a personas que ya sigue o a sí mismo.

   **Pregunta:** Este método de recomendar usuarios basado en amigos en común, ¿Qué tipo de relación implica?

## 3.2 Gimnasio

En este ejercicio, vamos a implementar una clase para gestionar las membresías y las operaciones de un gimnasio. Define tu propia estructura de proyecto usando módulos y paquetes.

Crea una clase que representará una **Membresía**. Esta clase debe tener los atributos necesarios para identificar una membresía y manejar sus operaciones básicas como son: su número de identificación único, la cuota mensual y el nombre del miembro asociado.

También, tendremos dos tipos diferentes de membresías:

1. **Membresía básica**: Esta membresía permite acceso limitado a las instalaciones (solo durante horas específicas del día) y no incluye clases grupales.
2. **Membresía premium**: Esta membresía permite acceso completo al gimnasio durante todo el día e incluye acceso a todas las clases grupales y entrenadores personales.

**Pregunta**: ¿Cuál es la relación existente entre los distintos tipos de membresías?

A continuación, crea una clase que represente a un **Miembro** del gimnasio. Cada miembro tiene un nombre y está asociado con una **Membresía** única para ese miembro en particular.

**Pregunta**: ¿Qué tipo de relación existe entre el miembro y la membresía?

Además, crea una clase que represente un **Instructor**. Un instructor en el gimnasio tiene un nombre, especialidades (como yoga, crossfit, etc.), y el horario de clases que imparte. Los miembros con una membresía premium podrán solicitar clases individuales con los instructores.

**Pregunta**: ¿Cuál es la relación entre un instructor y una membresía premium?

Implementa la siguiente funcionalidad:

1. **Gestión de membresías**
   - Crea un método para registrar un nuevo miembro en el gimnasio, lo que implicará crear una nueva membresía asociada a dicho miembro. Este miembro será añadido al sistema de gestión del gimnasio.
   - Crea un método que permita que un miembro actualice su membresía de básica a premium o viceversa. Este método deberá ajustar la cuota mensual y las condiciones de acceso a servicios en función del nuevo tipo de membresía.
2. **Clases y entrenamientos**

   Permite a los miembros con **membresía premium** agendar una clase privada con un instructor. El sistema debe comprobar la disponibilidad de la actividad antes de aceptar la solicitud.

   **Pregunta**: ¿Qué relaciones generan las clases solicitadas con instructores?
3. **Gestión del gimnasio**
   - Crea un método que devuelva una lista de todos los miembros registrados en el gimnasio junto con los detalles de sus tipos de membresía (básica o premium).
   - Crea un método que calculará los ingresos mensuales del gimnasio teniendo en cuenta las cuotas de todas las membresías activas.

## 3.3 Inventario de productos

En este ejercicio, vamos a implementar un sistema para gestionar el inventario de productos de una empresa. Define tu propia estructura de proyecto usando módulos y paquetes.

Crea una clase que representará un **Producto**. Esta clase debe tener los atributos necesarios para identificar un producto y manejar sus operaciones básicas, como son: su identificador único, el nombre del producto, el precio por unidad, la cantidad en stock y el proveedor asociado que suministra el producto.

También tendremos dos tipos diferentes de productos:

1. **Producto perecedero**: Este producto tiene una fecha de caducidad, y su cantidad en stock debe gestionarse para evitar el desperdicio.
2. **Producto no perecedero**: Este producto no tiene fecha de caducidad y puede mantenerse en stock indefinidamente.

**Pregunta**: ¿Cuál es la relación existente entre los distintos tipos de productos?

A continuación, crea una clase que represente un **Proveedor**. Cada proveedor tiene un nombre, un catálogo de productos que suministra y un contacto (por ejemplo, un número de teléfono o correo electrónico).

**Pregunta**: ¿Qué tipo de relación existe entre el proveedor y los productos?

Además, crea una clase que represente un **Almacén**. Un almacén tiene un nombre, una dirección y un inventario de productos que gestiona.

**Pregunta**: ¿Cuál es la relación entre un almacén y los productos que almacena?

Para gestionar la reposición de productos y mantener el stock adecuado, vamos a introducir el concepto de **Pedido**. Un pedido será realizado cuando el stock de un producto sea bajo y se necesite reponer. Un pedido se caracteriza por los siguientes atributos:

- El **producto** que se está solicitando reponer.
- La cantidad solicitada en el pedido.
- El **proveedor** al que se hace el pedido.
- La fecha de solicitud del pedido.

**Pregunta**: ¿Qué tipo de relación existe entre los pedidos y los productos? ¿Y entre los productos y los proveedores?

Implementa las siguientes funcionalidades:

1. **Gestión de productos**
   - Crea un método que permita registrar un nuevo producto en el inventario del almacén.

- Crea un método que permita modificar la cantidad en stock de un producto específico en el inventario del almacén.
- Crea un método que permita eliminar un producto del inventario cuando ya no esté disponible o haya sido descontinuado.

2. **Gestión de proveedores**
   - Asignar un proveedor a un producto: Crea un método que permita asignar un proveedor a un producto específico.
   - Consultar los productos de un proveedor: Crea un método que devuelva el catálogo de productos que suministra un proveedor.

   **Pregunta**: ¿Qué tipo de relación implica la consulta del catálogo de productos de un proveedor?
3. **Gestión de pedidos y ventas**
   - Crea un método que permita registrar una venta de un producto. El método debe actualizar la cantidad en stock y calcular el total de la venta (cantidad vendida * precio por unidad).
   - Crea un método que permita registrar un pedido de reposición de un producto cuando su cantidad en stock sea baja. El pedido deberá registrar la cantidad solicitada al proveedor, la fecha de solicitud y el producto al que está asociado.