Índice

Ejercicios Unidad 19 - Colecciones

Descargar estos ejercicios

> **Antes de empezar**
>
> Para realizar estos ejercicios, deberás descargar los recursos del enlace de proyecto_poo. Como puedes ver, la solución está compuesta de varios proyectos. Cada uno de ellos corresponde con un ejercicio, deberás implementar todo el código, tanto de la Main como de los métodos que se piden en cada ejercicio. Cada proyecto contiene el test correspondiente, que deberás pasar para comprobar que has hecho el ejercicio correctamente.

Ejercicio 1. Gestión de lista de automóviles

Implementa un sistema para gestionar una colección de automóviles utilizando `List<T>` y métodos para realizar operaciones básicas como añadir, eliminar, buscar y filtrar automóviles según diferentes criterios.

```
Ejercicio 1: Gestión de lista de automóviles

=== LISTA INICIAL DE AUTOMÓVILES ===
Añadiendo automóviles a la lista...
Lista actual (5 automóviles):
1. Toyota Corolla - 1600cc - 2020 - Blanco
2. Honda Civic - 1800cc - 2019 - Negro
3. Ford Focus - 2000cc - 2020 - Rojo
4. Nissan Sentra - 1600cc - 2018 - Azul
5. Volkswagen Golf - 1400cc - 2020 - Blanco

=== BÚSQUEDA POR AÑO DE FABRICACIÓN ===
Automóviles del año 2020:
- Toyota Corolla - 1600cc - 2020 - Blanco
- Ford Focus - 2000cc - 2020 - Rojo
- Volkswagen Golf - 1400cc - 2020 - Blanco

=== BÚSQUEDA POR AÑO Y COLOR ===
Automóviles del año 2020 y color Blanco:
- Toyota Corolla - 1600cc - 2020 - Blanco
- Volkswagen Golf - 1400cc - 2020 - Blanco

=== ELIMINACIÓN DE AUTOMÓVIL ===
Eliminando automóvil en posición 2 (Honda Civic)...
Lista actualizada (4 automóviles):
1. Toyota Corolla - 1600cc - 2020 - Blanco
2. Ford Focus - 2000cc - 2020 - Rojo
3. Nissan Sentra - 1600cc - 2018 - Azul
4. Volkswagen Golf - 1400cc - 2020 - Blanco

=== AÑADIR NUEVO AUTOMÓVIL ===
Añadiendo: BMW Serie 3 - 2000cc - 2021 - Gris
Lista final (5 automóviles):
1. Toyota Corolla - 1600cc - 2020 - Blanco
2. Ford Focus - 2000cc - 2020 - Rojo
3. Nissan Sentra - 1600cc - 2018 - Azul
4. Volkswagen Golf - 1400cc - 2020 - Blanco
5. BMW Serie 3 - 2000cc - 2021 - Gris

Fin de la demostración.
```

Requisitos

- Define un record `Automovil` con propiedades: `Marca`, `Modelo`, `Cilindrada` (int), `AñoFabricacion` (int), `Color`.
- Implementa un método `ToString()` que muestre la información del automóvil en formato como se muestra en la salida.
- Crea la clase `Program` con los siguientes métodos estáticos:
  - Método `AñadeAutomovil` que a partir de una lista y un automóvil, añadirá este a la lista.
  - `EliminaAutomovil` que eliminará el automóvil con la posición en la lista que coincida con el índice `i` que se haya pasado como argumento.
  - Crea un método `AutomovilesPorAñoFabricacion`, que te permita encontrar en la lista los coches con una determinada fecha de fabricación y que retorne una nueva lista con esos datos. ***Usa un bucle for para recorrer la lista y encontrar los automóviles que cumplan la condición.***
  - Otro método `AutomovilesPorAñoFabricacionYColor` que devuelva una sublista con los coches de la lista que sean de un determinado color y una fecha pasados ambos como parámetros. ***Usa un bucle for para recorrer la lista y encontrar los automóviles que cumplan la condición***.
  - `MostrarLista(List<Automovil> lista)` muestra el contenido completo de la lista numerado.
- Validaciones necesarias:
  - Verificar que el índice sea válido antes de eliminar.
  - Comprobar que la lista no sea null en todos los métodos.
  - Manejar casos donde no se encuentren automóviles que coincidan con los criterios de búsqueda.
- En el método `Main`, demuestra el uso de todos los métodos creando una lista inicial, realizando búsquedas, eliminaciones y adiciones.
- (Opcional) Añade métodos adicionales como `AutomovilesPorMarca(List<Automovil> lista, string marca)` o `AutomovilesOrdenadosPorAño(List<Automovil> lista)`.

Ejercicio 2. Sistema de pedidos de restaurante con

colecciones especializadas

Desarrolla un sistema para gestionar pedidos de un restaurante usando colecciones apropiadas para cada tipo de operación: menú ordenado, ingredientes únicos, cola de pedidos y histórico de clientes.

```
Ejercicio 2: Sistema de pedidos de restaurante

=== MENÚ DEL DÍA (por categorías) ===
ENTRANTES:
  - Ensalada César ($8.50)
  - Bruschetta ($6.00)
PRINCIPALES:
  - Paella Valenciana ($15.20)
  - Salmón a la plancha ($18.50)
POSTRES:
  - Tiramisu ($7.80)
  - Flan casero ($5.50)

=== GESTIÓN DE INGREDIENTES ===
Ingredientes disponibles: Lechuga, Tomate, Queso, Arroz, Azafrán, Salmón, Huevos, Café
Platos vegetarianos disponibles: Ensalada César, Flan casero
Alerta: El ingrediente 'Azafrán' está en varios platos (verificar stock)

=== COLA DE PEDIDOS ===
Cola actual (3 pedidos esperando):
1. Mesa 5: Paella Valenciana, Tiramisu - Total: $23.00
2. Mesa 2: Ensalada César, Salmón a la plancha - Total: $27.00
3. Mesa 8: Bruschetta, Flan casero - Total: $11.50

Cocinando pedido de Mesa 5...
Pedido completado. Cola actualizada (2 pedidos).

=== CLIENTES FRECUENTES ===
Top 3 clientes por número de visitas:
1. Juan Martínez - 15 visitas - Gasto promedio: $25.30
2. Ana López - 12 visitas - Gasto promedio: $31.50
3. Carlos García - 8 visitas - Gasto promedio: $19.80

Cliente Juan Martínez - Historial últimas 3 visitas:
- 2025-10-05: $28.50 (Paella + Tiramisu)
- 2025-10-02: $22.10 (Ensalada + Salmón)
- 2025-09-28: $15.20 (Solo Paella)

Fin de la demostración.
```

Requisitos

- Clase `Plato`: `Nombre`, `Precio`, `Categoria` (enum: Entrante, Principal, Postre), `Ingredientes` (HashSet).
- Clase `Pedido`: `NumeroMesa`, `Platos` (List), `FechaHora`, `Total` (calculado).
- Clase `Cliente`: `Nombre`, `HistorialPedidos` (List), propiedades calculadas: `NumeroVisitas`, `GastoPromedio`.
- Clase `Restaurante` que use:
  - `SortedDictionary<Categoria, List<Plato>>` para menú organizado.
  - `HashSet<string>` para control de ingredientes disponibles.
  - `Queue<Pedido>` para cola de cocina.
  - `Dictionary<string, Cliente>` para clientes frecuentes.
- Métodos principales:
  - `AgregarPlato(Plato plato)` - organiza por categoría.
  - `PlatosVegetarianos()` - filtra platos sin ingredientes de origen animal.
  - `EncolarPedido(Pedido pedido)` - añade a cola de cocina.
  - `ProcesarPedido()` - saca de la cola y procesa.
  - `ActualizarCliente(string nombre, Pedido pedido)` - actualiza historial.
  - `TopClientesFrecuentes(int cantidad)` - ordena por número de visitas.
- Validaciones: verificar ingredientes disponibles antes de aceptar pedidos.
- (Opcional) Implementa sistema de descuentos para clientes frecuentes.

Ejercicio 3. Red social simplificada con grafos y colecciones anidadas

Implementa una red social básica que permita gestionar usuarios, conexiones (amistades) y publicaciones usando colecciones que representen eficientemente las relaciones entre usuarios.

```
Ejercicio 3: Red social simplificada

=== USUARIOS REGISTRADOS ===
- juan.perez (Juan Pérez) - 5 amigos, 12 publicaciones
- ana.garcia (Ana García) - 8 amigos, 7 publicaciones
- carlos.ruiz (Carlos Ruiz) - 3 amigos, 15 publicaciones
- maria.lopez (María López) - 6 amigos, 9 publicaciones
- luis.martin (Luis Martín) - 4 amigos, 11 publicaciones

=== RED DE CONEXIONES ===
Amigos de juan.perez: ana.garcia, carlos.ruiz, maria.lopez
Amigos en común entre juan.perez y ana.garcia: carlos.ruiz, maria.lopez
Usuarios conectados por un enlace desde juan.perez: ana.garcia, carlos.ruiz, maria.lope

=== FEED DE NOTICIAS (juan.perez) ===
[Ana García - hace 2h] "¡Día perfecto para programar! 💻"
  ❤️ 3 likes - 1 comentario
[Carlos Ruiz - hace 4h] "Nueva funcionalidad implementada en el proyecto"
  ❤️ 7 likes - 3 comentarios
[María López - hace 6h] "Compartiendo conocimiento en el equipo"
  ❤️ 2 likes - 0 comentarios

=== ESTADÍSTICAS DE LA RED ===
Total usuarios: 5
Total conexiones: 13
Usuario más conectado: ana.garcia (8 amigos)
Usuario más activo: carlos.ruiz (15 publicaciones)
Grado de separación promedio: 2.1

=== RECOMENDACIONES ===
Personas que juan.perez podría conocer:
- luis.martin (2 amigos en común: ana.garcia, maria.lopez)

Publicaciones trending (más likes últimas 24h):
1. "Nueva funcionalidad implementada..." - carlos.ruiz (7 likes)
2. "¡Día perfecto para programar!..." - ana.garcia (3 likes)

Fin de la demostración.
```

Requisitos

- Clase `Usuario`: `Username`, `NombreCompleto`, `FechaRegistro`.
- Clase `Publicacion`: `Id`, `Autor`, `Contenido`, `FechaHora`, `Likes` (HashSet), `Comentarios` (List).
- Clase `RedSocial` que use:
  - `Dictionary<string, Usuario>` para usuarios registrados.
  - `Dictionary<string, HashSet<string>>` para grafo de amistades (adyacencias).
  - `Dictionary<string, List<Publicacion>>` para publicaciones por usuario.

    - `SortedSet<Publicacion>` para feed ordenado por fecha (implementar IComparable).
- Métodos principales:
    - `RegistrarUsuario(Usuario usuario)` - añade usuario al sistema.
    - `ConectarUsuarios(string user1, string user2)` - establece amistad bidireccional.
    - `AmigosEnComun(string user1, string user2)` - encuentra intersección.
    - `AmigosDeAmigos(string username)` - encuentra usuarios a distancia 2.
    - `GenerarFeed(string username)` - obtiene publicaciones de amigos ordenadas.
    - `RecomendarAmigos(string username)` - sugiere basado en amigos en común.
    - `PublicacionesTrending(int horas)` - más populares en período de tiempo.
- Algoritmos de grafos:
    - Búsqueda en anchura (BFS) para calcular distancias.
    - Cálculo de centralidad simple (número de conexiones).
- Validaciones: evitar autoconexiones, conexiones duplicadas.
- (Opcional) Implementa algoritmo para detectar comunidades usando clustering simple.

Ejercicio 4.

Ejercicio 5.

Ejercicio 6.