Exercices Aprog 4ème

LINQ - Session 12

21 janvier 2026

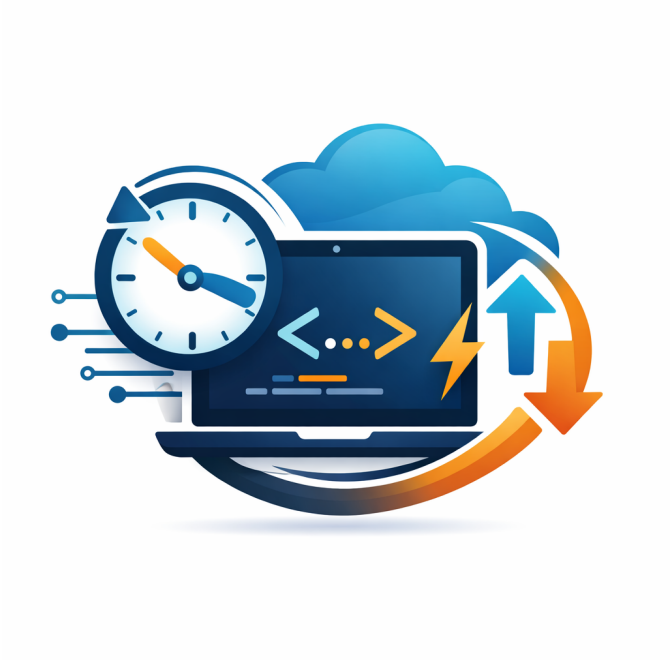

Table des matières

1 Exercices - LINQ (Language Integrated Query)

1.1 Consignes générales importantes

Architecture et séparation des responsabilités:

Pour tous les exercices, vous devez respecter les principes suivants:

1. **Séparation des responsabilités** - Séparez votre code en classes distinctes selon leur rôle:
   - **Models** (Modèles): Classes représentant les données (ex: `Personne`, `Produit`, `Commande`)
   - **Services**: Classes contenant la logique métier et les requêtes LINQ (ex: `PersonneService`, `ProduitService`)
   - **Views** (Vues): Classes gérant l'affichage et l'interaction avec l'utilisateur (ex: `ConsoleView`, `MenuView`)
   - **Program**: Point d'entrée de l'application qui coordonne les autres classes
2. **Utilisation correcte des classes et accessibilités:**
   - Utilisez les modificateurs d'accès appropriés: `public`, `private`, `protected`, `internal`
   - Les propriétés des modèles doivent utiliser `{ get; set; }` ou `{ get; init; }`
   - Les méthodes de service doivent être `public` si utilisées de l'extérieur
   - Les champs privés doivent commencer par _ (ex: `private List<Personne> _personnes`)
3. **Architecture minimale Model-View:**
   - Au minimum, séparez les **Models** (données) et les **Views** (affichage)
   - Idéalement, ajoutez une couche **Service** pour la logique métier

Exemple de structure de projet:

```
Projet/
  Models/
    Personne.cs
  Services/
    PersonneService.cs
  Views/
    ConsoleView.cs
  Program.cs
```

1.2 Exercice 1: Filtrage de base avec Where

Objectif: Maîtriser l'opérateur `Where` pour filtrer des collections.

Consignes:

1. **Créez les classes suivantes:**
   - `Etudiant` (Model): propriétés `Id`, `Nom`, `Prenom`, `Age`, `Moyenne`
   - `EtudiantService` (Service):
     - Champ `private List<Etudiant> _etudiants`
     - Méthode `public List<Etudiant> ObtenirEtudiantsMajeurs()`
     - Méthode `public List<Etudiant> ObtenirEtudiantsAvecMoyenneSuperieure(double seuilMoyenne)`
     - Méthode `public List<Etudiant> ObtenirEtudiantsParTrancheAge(int ageMin, int ageMax)`
   - `EtudiantView` (View):
     - Méthode `public void AfficherListeEtudiants(List<Etudiant> etudiants, string titre)`
2. Créez une liste d'au moins 10 étudiants avec des données variées
3. Dans le `Main`:
   - Instanciez le service et la vue
   - Affichez les étudiants majeurs (18 ans et plus)
   - Affichez les étudiants avec une moyenne supérieure à 12
   - Affichez les étudiants entre 20 et 25 ans

Données suggérées:

```
new Etudiant { Id = 1, Nom = "Dupont", Prenom = "Jean", Age = 22, Moyenne = 14.5 }
new Etudiant { Id = 2, Nom = "Martin", Prenom = "Marie", Age = 17, Moyenne = 16.0 }
// ... ajoutez au moins 8 autres étudiants
```

Bonus: Ajoutez une méthode qui filtre les étudiants dont le nom commence par une lettre donnée.

1.3 Exercice 2: Tri avec OrderBy et ThenBy

Objectif: Maîtriser le tri simple et multiple avec LINQ.

Consignes:

1. **Créez les classes suivantes:**
   - `Employe` (Model): propriétés `Id`, `Nom`, `Prenom`, `Departement`, `Salaire`, `DateEmbauche`
   - `EmployeService` (Service):
     - Champ `private List<Employe> _employes`
     - Méthode `public IOrderedEnumerable<Employe> TrierParSalaireCroissant()`
     - Méthode `public IOrderedEnumerable<Employe> TrierParSalaireDecroissant()`
     - Méthode `public IOrderedEnumerable<Employe> TrierParDepartementPuisNom()`
     - Méthode `public IOrderedEnumerable<Employe> TrierParAnciennete()`
   - `EmployeView` (View):
     - Méthode `public void AfficherTableauEmployes(IEnumerable<Employe> employes, string titre)`
2. Créez une liste d'au moins 10 employés répartis dans 3 départements différents
3. Dans le `Main`:
   - Testez tous les tris et affichez les résultats
   - Comparez les ordres croissant et décroissant

Départements suggérés: "Informatique", "Comptabilité", "Marketing"

Bonus: Ajoutez un tri par département (croissant), puis par salaire (décroissant) au sein de chaque département.

1.4 Exercice 3: Projection avec Select

Objectif: Transformer des données avec l'opérateur `Select`.

Consignes:

1. **Créez les classes suivantes:**
   - `Produit` (Model): propriétés `Id`, `Nom`, `PrixHT`, `TauxTVA`, `Categorie`, `Stock`
   - `ProduitAffichage` (Model): propriétés `NomComplet`, `PrixTTC`, `Disponibilite` (pour la projection)
   - `ProduitService` (Service):
     - Champ `private List<Produit> _produits`
     - Méthode `public IEnumerable<string> ObtenirNomsEnMajuscules()`
     - Méthode `public IEnumerable<ProduitAffichage> ObtenirProduitsAvecPrixTTC()`
     - Méthode `public IEnumerable<string> ObtenirDescriptionsFormatees()`
   - `ProduitView` (View):
     - Méthode `public void AfficherNoms(IEnumerable<string> noms)`
     - Méthode `public void AfficherProduitsFormates(IEnumerable<ProduitAffichage> produits)`
2. Créez une liste de produits avec différentes catégories et taux de TVA
3. Dans le `Main`:
   - Affichez uniquement les noms en majuscules
   - Affichez les produits avec leur prix TTC calculé
   - Affichez une description formatée: "Produit X - CHF Y.YY (Stock: Z)"

Formule TVA: `PrixTTC = PrixHT * (1 + TauxTVA / 100)`

Bonus: Créez une projection vers un type anonyme contenant le nom et le prix TTC arrondi à 2 décimales.

1.5 Exercice 4: Agrégation avec Count, Sum, Average, Min, Max

Objectif: Calculer des statistiques sur des collections.

Consignes:

1. **Créez les classes suivantes:**
   - `Vente` (Model): propriétés `Id`, `DateVente`, `Montant`, `Vendeur`, `Region`
   - `StatistiquesVente` (Model): propriétés `NombreVentes`, `MontantTotal`, `MontantMoyen`, `MontantMin`, `MontantMax`
   - `VenteService` (Service):
     - Champ `private List<Vente> _ventes`
     - Méthode `public StatistiquesVente CalculerStatistiquesGlobales()`
     - Méthode `public StatistiquesVente CalculerStatistiquesParVendeur(string vendeur)`
     - Méthode `public int CompterVentesSuperieurA(decimal seuil)`
     - Méthode `public decimal CalculerChiffreAffairesMois(int mois, int annee)`
   - `StatistiquesView` (View):
     - Méthode `public void AfficherStatistiques(StatistiquesVente stats, string titre)`
2. Créez une liste de ventes sur plusieurs mois avec différents vendeurs
3. Dans le `Main`:
   - Affichez les statistiques globales
   - Affichez les statistiques pour un vendeur spécifique
   - Comptez les ventes supérieures à 500 CHF

Bonus: Calculez le vendeur avec le meilleur chiffre d'affaires total.

1.6 Exercice 5: Regroupement avec GroupBy

Objectif: Regrouper des données et calculer des agrégats par groupe.

Consignes:

1. **Créez les classes suivantes:**
   - `Commande` (Model): propriétés `Id`, `Client`, `Produit`, `Quantite`, `PrixUnitaire`, `DateCommande`
   - `ResumeParClient` (Model): propriétés `NomClient`, `NombreCommandes`, `MontantTotal`
   - `CommandeService` (Service):
     - Champ `private List<Commande> _commandes`
     - Méthode `public IEnumerable<IGrouping<string, Commande>> GrouperParClient()`
     - Méthode `public IEnumerable<ResumeParClient> ObtenirResumeParClient()`
     - Méthode `public IEnumerable<IGrouping<int, Commande>> GrouperParMois()`
   - `CommandeView` (View):
     - Méthode `public void AfficherGroupesParClient(IEnumerable<IGrouping<string, Commande>> groupes)`
     - Méthode `public void AfficherResumes(IEnumerable<ResumeParClient> resumes)`
2. Créez une liste de commandes avec plusieurs clients et produits
3. Dans le `Main`:
   - Groupez les commandes par client et affichez chaque groupe
   - Affichez le résumé par client (nombre de commandes et montant total)
   - Groupez par mois et affichez le chiffre d'affaires mensuel

Calcul montant: `Quantite * PrixUnitaire`

Bonus: Trouvez le client ayant passé le plus de commandes.

1.7 Exercice 6: Opérations ensemblistes (Distinct, Union, Intersect, Except)

Objectif: Manipuler des ensembles avec LINQ.

Consignes:

1. **Créez les classes suivantes:**
   - `Competence` (Model): propriétés `Id`, `Nom`, `Niveau` (de 1 à 5)
   - `Candidat` (Model): propriétés `Id`, `Nom`, `Competences` (List)
   - `OffreEmploi` (Model): propriétés `Id`, `Titre`, `CompetencesRequises` (List)
   - `RecrutementService` (Service):
     - Méthode `public IEnumerable<string> ObtenirCompetencesUniques(List<Candidat> candidats)`
     - Méthode `public IEnumerable<string> ObtenirCompetencesCommunes(Candidat candidat1, Candidat candidat2)`
     - Méthode `public IEnumerable<string> ObtenirCompetencesManquantes(Candidat candidat, OffreEmploi offre)`
     - Méthode `public IEnumerable<string> ObtenirToutesCompetences(Candidat candidat1, Candidat candidat2)`
   - `RecrutementView` (View):
     - Méthode `public void AfficherCompetences(IEnumerable<string> competences, string titre)`
2. Créez plusieurs candidats avec des compétences qui se chevauchent partiellement
3. Dans le `Main`:
   - Affichez toutes les compétences uniques (sans doublons) de tous les candidats
   - Comparez deux candidats: compétences communes et différences
   - Vérifiez quelles compétences manquent à un candidat pour une offre

Compétences suggérées: “C#”, “SQL”, “JavaScript”, “Python”, “Git”, “Azure”, “Docker”

Bonus: Trouvez le candidat le plus qualifié pour une offre (celui avec le plus de compétences correspondantes).

1.8 Exercice 7: Sous-ensembles avec Take, Skip et pagination

Objectif: Extraire des portions de collections et implémenter la pagination.

Consignes:

1. **Créez les classes suivantes:**
   - `Article` (Model): propriétés `Id`, `Titre`, `Contenu`, `DatePublication`, `Auteur`, `NombreVues`
   - `PageResultat<T>` (Model): propriétés `Elements`, `PageActuelle`, `TotalPages`, `TotalElements`
   - `ArticleService` (Service):
     - Champ `private List<Article> _articles`
     - Méthode `public IEnumerable<Article> ObtenirTopArticles(int nombre)`
     - Méthode `public IEnumerable<Article> ObtenirArticlesRecents(int nombre)`
     - Méthode `public PageResultat<Article> ObtenirPage(int numeroPage, int tailePage)`
     - Méthode `public IEnumerable<Article> IgnorerPremiers(int nombre)`
   - `ArticleView` (View):
     - Méthode `public void AfficherArticles(IEnumerable<Article> articles, string titre)`
     - Méthode `public void AfficherPage(PageResultat<Article> page)`
2. Créez une liste d’au moins 25 articles
3. Dans le `Main`:
   - Affichez les 5 articles les plus vus
   - Affichez les 10 articles les plus récents
   - Implémentez une pagination de 5 articles par page et affichez la page 2

Bonus: Ajoutez une navigation interactive permettant de passer d’une page à l’autre.

1.9 Exercice 8: Vérifications avec Any, All et Contains

Objectif: Vérifier des conditions sur des collections.

Consignes:

1. **Créez les classes suivantes:**
   - `Tache` (Model): propriétés `Id`, `Titre`, `Description`, `EstTerminee`, `Priorite` (1-5), `DateEcheance`
   - `ResultatValidation` (Model): propriétés `EstValide`, `Messages` (List)
   - `TacheService` (Service):
     - Champ `private List<Tache> _taches`
     - Méthode `public bool ExisteTachesEnRetard()`
     - Méthode `public bool ToutesTachesTerminees()`
     - Méthode `public bool ExisteTacheAvecPriorite(int priorite)`
     - Méthode `public ResultatValidation ValiderListeTaches()`
   - `TacheView` (View):
     - Méthode `public void AfficherResultatValidation(ResultatValidation resultat)`
     - Méthode `public void AfficherEtatProjet(bool tachesEnRetard, bool toutTermine)`
2. Créez une liste de tâches avec différents états et dates d'échéance
3. Dans le `Main`:
   - Vérifiez s'il existe des tâches en retard (date échéance passée et non terminée)
   - Vérifiez si toutes les tâches sont terminées
   - Vérifiez si une tâche de priorité 5 (urgente) existe

Bonus: Créez une méthode qui vérifie si toutes les tâches urgentes sont terminées.

1.10 Exercice 9: Requêtes combinées

Objectif: Combiner plusieurs opérateurs LINQ dans des requêtes complexes.

Consignes:

1. **Créez les classes suivantes:**
   - `Film` (Model): propriétés `Id`, `Titre`, `Annee`, `Genre`, `Note`, `Duree`, `Realisateur`
   - `StatistiquesGenre` (Model): propriétés `Genre`, `NombreFilms`, `NoteMoyenne`, `MeilleurFilm`
   - `FilmService` (Service):
     - Champ `private List<Film> _films`
     - Méthode `public IEnumerable<Film> RechercherFilms(string genre, int? anneeMin, double? noteMin)`
     - Méthode `public IEnumerable<StatistiquesGenre> ObtenirStatistiquesParGenre()`
     - Méthode `public IEnumerable<Film> ObtenirTopFilmsParGenre(string genre, int nombre)`
     - Méthode `public IEnumerable<string> ObtenirRealisateursProlifiques(int nombreFilmsMin)`
   - `FilmView` (View):
     - Méthode `public void AfficherFilms(IEnumerable<Film> films)`
     - Méthode `public void AfficherStatistiquesGenre(IEnumerable<StatistiquesGenre> stats)`
2. Créez une liste d'au moins 20 films de différents genres
3. Dans le `Main`:
   - Recherchez les films d'action sortis après 2010 avec une note > 7
   - Affichez les statistiques par genre
   - Affichez le top 3 des films par genre
   - Listez les réalisateurs ayant réalisé au moins 2 films

Genres suggérés: "Action", "Comédie", "Drame", "Science-Fiction", "Horreur"

Bonus: Trouvez l'année avec le plus de films sortis.

1.11 Exercice 10: Projet final - Gestionnaire de bibliothèque

Objectif: Créer une application complète utilisant tous les concepts LINQ.

Consignes:

1. **Créez une architecture complète Model-View-Service:**
   Models/
   - `Livre`: propriétés `Id`, `Titre`, `Auteur`, `ISBN`, `Annee`, `Genre`, `NombrePages`, `EstDisponible`
   - `Emprunt`: propriétés `Id`, `LivreId`, `NomEmprunteur`, `DateEmprunt`, `DateRetourPrevue`, `DateRetourEffective`
   - `StatistiquesBibliotheque`: propriétés `TotalLivres`, `LivresDisponibles`, `LivresEmpruntes`, `EmpruntsEnRetard`
   - `RapportGenre`: propriétés `Genre`, `NombreLivres`, `NombreEmprunts`, `LivreLePlusEmprunte`

   Services/
   - `LivreService`:
     - Champ `private List<Livre> _livres`
     - Méthode `public IEnumerable<Livre> RechercherLivres(string motCle)`
     - Méthode `public IEnumerable<Livre> FiltrerParGenre(string genre)`
     - Méthode `public IEnumerable<Livre> ObtenirLivresDisponibles()`
     - Méthode `public IEnumerable<IGrouping<string, Livre>> GrouperParAuteur()`
   - `EmpruntService`:
     - Champ `private List<Emprunt> _emprunts`
     - Méthode `public IEnumerable<Emprunt> ObtenirEmpruntsEnCours()`
     - Méthode `public IEnumerable<Emprunt> ObtenirEmpruntsEnRetard()`
     - Méthode `public IEnumerable<string> ObtenirMeilleursEmprunteurs(int top)`
   - `StatistiquesService`:
     - Méthode `public StatistiquesBibliotheque CalculerStatistiques(List<Livre> livres, List<Emprunt> emprunts)`
     - Méthode `public IEnumerable<RapportGenre> GenererRapportParGenre(List<Livre> livres, List<Emprunt> emprunts)`

   Views/
   - `BibliothequeView`:
     - Méthode `public int AfficherMenuPrincipal()`
     - Méthode `public void AfficherLivres(IEnumerable<Livre> livres)`
     - Méthode `public void AfficherEmprunts(IEnumerable<Emprunt> emprunts)`
     - Méthode `public void AfficherStatistiques(StatistiquesBibliotheque stats)`
     - Méthode `public void AfficherRapportGenre(IEnumerable<RapportGenre> rapport)`
     - Méthode `public string DemanderRecherche()`
2. **Fonctionnalités à implémenter:**
   - Recherche de livres par titre, auteur ou ISBN
   - Filtrage par genre et disponibilité
   - Affichage des emprunts en cours et en retard
   - Statistiques globales de la bibliothèque
   - Rapport détaillé par genre
   - Top des emprunteurs
3. **Menu interactif:**

```
=== Gestionnaire de Bibliothèque ===
1. Rechercher un livre
2. Voir les livres disponibles
3. Voir les livres par genre
4. Voir les emprunts en cours
5. Voir les emprunts en retard
6. Afficher les statistiques
7. Générer rapport par genre
8. Top emprunteurs
9. Quitter
```

Architecture attendue:

```
Projet/
  Models/
    Livre.cs
    Emprunt.cs
    StatistiquesBibliotheque.cs
    RapportGenre.cs
  Services/
    LivreService.cs
    EmpruntService.cs
    StatistiquesService.cs
  Views/
    BibliothequeView.cs
  Program.cs
```

Données initiales suggérées:

- Au moins 15 livres de genres variés
- Au moins 10 emprunts (certains en cours, certains en retard, certains rendus)

Bonus:

- Ajoutez une fonctionnalité d'export des statistiques en fichier texte
- Implémentez une recherche avec plusieurs critères combinés (genre ET disponibilité ET année)