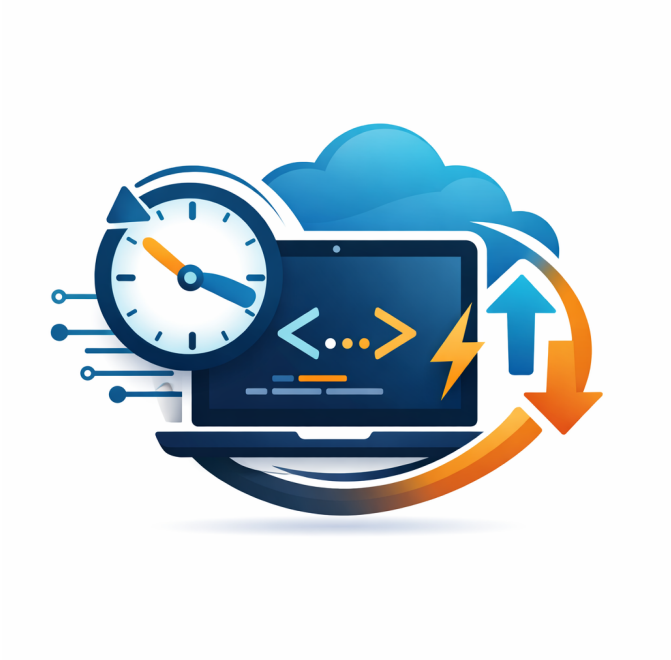
# Exercices Aprog 4ème

Async - Await - Session 11

14 janvier 2026



# Table des matières

# 1 Exercices - Programmation Asynchrone avec async/await

## 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: `Utilisateur`, `Produit`, `Meteo`)
   - **Services**: Classes contenant la logique métier et les appels asynchrones (ex: `UtilisateurService`, `ProduitRepository`)
   - **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 HttpClient _client`)
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/
    Utilisateur.cs
  Services/
    UtilisateurService.cs
  Views/
    ConsoleView.cs
  Program.cs
```

---

## 1.2 Exercice 1: Téléchargement de fichiers depuis Internet

**Objectif:** Créer une application console qui télécharge plusieurs images depuis Internet de manière asynchrone.

**Consignes:**

1. **Créez les classes suivantes:**
   - `ImageTelechargement` (Model): propriétés `Url`, `NomFichier`, `Statut`, `TailleOctets`
   - `ImageService` (Service): contient la méthode `TelechargerImageAsync(string url, string nomFichier)`
   - `ConsoleView` (View): méthode `AfficherResultat(ImageTelechargement image)` pour afficher les résultats
2. Dans votre méthode `Main`:
   - Créez une instance de `ImageService` et `ConsoleView`
   - Créez une liste d'URLs d'images (au moins 3)
   - Téléchargez toutes les images via le service
   - Utilisez la vue pour afficher les résultats
   - Affichez le temps total nécessaire

**URLs suggérées pour tester:**

- https://picsum.photos/800/600?random=1
- https://picsum.photos/800/600?random=2
- https://picsum.photos/800/600?random=3

**Bonus:** Gérez les erreurs de téléchargement avec un bloc try-catch et continuez le téléchargement des autres images même si l'une échoue.

---

## 1.3 Exercice 2: Appels API REST

**Objectif:** Créer une application qui consomme l'API publique JSONPlaceholder pour gérer des utilisateurs.

**Consignes:**

1. **Créez les classes suivantes:**
   - `Utilisateur` (Model): propriétés `Id`, `Name`, `Email`, `Phone` avec accessibilité appropriée
   - `UtilisateurService` (Service):
     - Champ privé `private readonly HttpClient _client`
     - Méthode `public async Task<Utilisateur> ObtenirUtilisateurAsync(int id)`
     - Méthode `public async Task<List<Utilisateur>> ObtenirTousUtilisateursAsync()`
   - `UtilisateurView` (View):
     - Méthode `public void AfficherUtilisateur(Utilisateur utilisateur)`
     - Méthode `public void AfficherListeUtilisateurs(List<Utilisateur> utilisateurs)`
2. Dans le `Main`:
   - Instanciez le service et la vue
   - Récupérez l'utilisateur avec l'ID 1
   - Récupérez la liste complète (limitez à 5 pour l'affichage)
   - Utilisez la vue pour afficher les résultats

**API à utiliser:** https://jsonplaceholder.typicode.com/users

**Bonus:** Ajoutez une méthode pour rechercher un utilisateur par nom.

---

## 1.4 Exercice 3: Lecture et écriture de fichiers

**Objectif:** Créer un programme qui traite des fichiers texte de manière asynchrone.

**Consignes:**

1. **Créez les classes suivantes:**
   - `StatistiquesTexte` (Model): propriétés `NombreMots`, `NombreLignes`, `TexteTraite`
   - `FichierService` (Service):
     - Méthode `public async Task<string> LireFichierAsync(string chemin)`
     - Méthode `public async Task<StatistiquesTexte> TraiterTexteAsync(string texte)`
     - Méthode `public async Task EcrireFichierAsync(string chemin, string contenu)`
   - `FichierView` (View):
     - Méthode `public void AfficherStatistiques(StatistiquesTexte stats)`
2. Créez un fichier `input.txt` avec plusieurs lignes de texte
3. Dans le `Main`:
   - Instanciez le service et la vue
   - Lisez le fichier via le service
   - Traitez le texte et obtenez les statistiques
   - Sauvegardez le résultat dans `output.txt`
   - Affichez les statistiques via la vue

**Bonus:** Ajoutez une fonctionnalité pour remplacer tous les espaces par des underscores.

---

## 1.5 Exercice 4: Exécution de tâches en parallèle

**Objectif:** Télécharger plusieurs pages web en parallèle et comparer avec un téléchargement séquentiel.

**Consignes:**

1. **Créez les classes suivantes:**
   - `ResultatTelechargement` (Model): propriétés `Url`, `TailleCaracteres`, `Duree`, `Succes`
   - `WebService` (Service):
     - Champ `private readonly HttpClient _client`
     - Méthode `public async Task<ResultatTelechargement> TelechargerPageAsync(string url)`
     - Méthode `public async Task<List<ResultatTelechargement>> TelechargerSequentiellementAsync(List<string> urls)`
     - Méthode `public async Task<List<ResultatTelechargement>> TelechargerEnParalleleAsync(List<string> urls)`
   - `PerformanceView` (View):
     - Méthode `public void AfficherComparaison(List<ResultatTelechargement> sequentiel, List<ResultatTelechargement> pa`
2. Dans le `Main`:
   - Testez les deux approches et mesurez les temps
   - Utilisez la vue pour afficher la comparaison

**URLs suggérées:** - https://example.com - https://httpbin.org/html - https://jsonplaceholder.typicode.com/posts - https://www.ietf.org

**Bonus:** Utilisez `Task.WhenAny` pour afficher les résultats au fur et à mesure qu'ils arrivent.

---

## 1.6 Exercice 5: Simulation d'opérations de base de données

**Objectif:** Simuler des opérations de base de données asynchrones avec des délais.

**Consignes:**

1. **Créez les classes suivantes:**
   - `Produit` (Model):
     - Propriétés `public int Id { get; init; }`, `public string Nom { get; set; }`, `public decimal Prix { get; set; }`, `public int Stock { get; set; }`
   - `ProduitRepository` (Service):
     - Champ `private static List<Produit> _produits` pour simuler une BD
     - Méthode `public async Task<List<Produit>> ObtenirTousProduitsAsync()`
     - Méthode `public async Task<Produit> ObtenirProduitParIdAsync(int id)`
     - Méthode `public async Task AjouterProduitAsync(Produit produit)`
     - Méthode `public async Task MettreAJourStockAsync(int id, int nouveauStock)`
     - Utilisez `Task.Delay` pour simuler les opérations BD
   - `ProduitView` (View):
     - Méthode `public void AfficherProduit(Produit produit)`
     - Méthode `public void AfficherListeProduits(List<Produit> produits)`
2. Dans le `Main`:
   - Testez toutes les opérations du repository
   - Utilisez la vue pour afficher les résultats

**Bonus:** Ajoutez une méthode `RechercherProduitsAsync(string motCle)` qui recherche par nom.

---

## 1.7 Exercice 6: Gestion d'erreurs avec async/await

**Objectif:** Créer un téléchargeur robuste qui gère les erreurs réseau.

**Consignes:**

1. **Créez les classes suivantes:**
   - `ResultatTentative` (Model): propriétés `Tentative`, `Succes`, `Message`, `Duree`
   - `TelechargementService` (Service):
     - Champ `private readonly HttpClient _client`
     - Méthode `public async Task<string> TelechargerAvecRetryAsync(string url, int maxTentatives, List<ResultatTentativ`
     - Gestion des exceptions: `HttpRequestException`, `TaskCanceledException`, `Exception`
   - `TelechargementView` (View):
     - Méthode `public void AfficherTentative(ResultatTentative tentative)`
     - Méthode `public void AfficherResultatFinal(bool succes, List<ResultatTentative> historique)`
2. Dans le `Main`:
   - Testez avec une URL valide et une URL invalide
   - Affichez les tentatives et le résultat final via la vue

**URLs pour tester:**

- URL valide: https://example.com
- URL invalide: https://urlquinexistepas123456.com

**Bonus:** Utilisez un délai exponentiel (2s, 4s, 8s) entre les tentatives.

---

## 1.8 Exercice 7: Annulation de tâches avec CancellationToken

**Objectif:** Créer une application qui permet d'annuler une opération longue.

**Consignes:**

1. **Créez les classes suivantes:**
   - `EtatTelechargement` (Model): propriétés `Progression`, `Statut`, `Message`
   - `FichierService` (Service):
     - Méthode `public async Task<bool> TelechargerGrossFichierAsync(string url, CancellationToken token, IProgress<Etat`
     - Simule 10 étapes avec `Task.Delay(1000, token)`
     - Vérifie `token.ThrowIfCancellationRequested()` à chaque étape
   - `ProgressionView` (View):
     - Méthode `public void AfficherProgression(EtatTelechargement etat)`
     - Méthode `public void AfficherAnnulation()`
     - Méthode `public void AfficherCompletion()`
2. Dans le `Main`:
   - Créez un `CancellationTokenSource` qui annule après 5 secondes
   - Gérez `OperationCanceledException`
   - Utilisez la vue pour tous les affichages

**Bonus:** Permettez à l'utilisateur d'annuler en appuyant sur une touche (utilisez `Console.KeyAvailable`).

---

## 1.9 Exercice 8: Progression avec IProgress

**Objectif:** Créer un téléchargeur qui rapporte sa progression en temps réel.

**Consignes:**

1. **Créez les classes suivantes:**
   - `InfoProgression` (Model): propriétés `Pourcentage`, `EtapeCourante`, `EtapesTotal`, `TempsEcoule`
   - `TraitementService` (Service):
     - Méthode `public async Task TraiterDonneesAvecProgressionAsync(int nombreEtapes, IProgress<InfoProgression> progre`
     - Simule chaque étape avec `Task.Delay(500)`
   - `ProgressionView` (View):
     - Méthode `public void AfficherBarreProgression(InfoProgression info)`
     - Affiche une barre visuelle: `[=========> ] 40%`
     - Méthode `public void AfficherCompletion(TimeSpan duree)`
2. Dans le `Main`:
   - Créez un `Progress<InfoProgression>` qui appelle la vue
   - Testez avec différents nombres d'étapes (10, 20, 50)

**Bonus:** Ajoutez un affichage du temps restant estimé.

---

## 1.10 Exercice 9: Application complète - Gestionnaire de téléchargements

**Objectif:** Créer une application console complète qui combine plusieurs concepts.

**Consignes:**

1. **Créez les classes suivantes:**
   - `Telechargement` (Model): propriétés `Url`, `NomFichier`, `Taille`, `DateHeure`, `Statut`
   - `TelechargementService` (Service):
     - Champ `private readonly HttpClient _client`
     - Champ `private List<Telechargement> _historique`
     - Méthode `public async Task<bool> TelechargerFichierAsync(string url, string nomFichier, IProgress<int> progress,`
     - Méthode `public async Task TelechargerPlusieursAsync(List<string> urls, IProgress<int> progress)`
     - Méthode `public async Task SauvegarderHistoriqueAsync(string chemin)`
     - Méthode `public async Task<List<Telechargement>> ChargerHistoriqueAsync(string chemin)`
   - `MenuView` (View):
     - Méthode `public int AfficherMenuPrincipal()`
     - Méthode `public void AfficherHistorique(List<Telechargement> historique)`
     - Méthode `public void AfficherProgression(int pourcentage)`
     - Méthode `public void AfficherUrlsSuggerees()` - Affiche des URLs de test
   - `Application` (Controller/Coordinateur):
     - Coordonne le service et la vue
     - Gère la boucle du menu
2. Implémentez toutes les fonctionnalités avec gestion d'erreurs et annulation
3. Architecture complète Model-View-Service requise

**URLs suggérées pour les tests:** - `https://example.com` - `https://httpbin.org/html` - `https://jsonplaceholder.typicode.com/posts` - `https://www.ietf.org`

**Bonus:** Ajoutez une limite de téléchargements simultanés (max 3) avec `SemaphoreSlim`.

---

## 1.11 Exercice 10: Projet final - Agrégateur de données météo

**Objectif:** Créer une application qui récupère les données météo de plusieurs villes en parallèle.

**Consignes:**

1. **Créez une architecture complète Model-View-Service:**
   **Models/**
   - `Meteo`: propriétés `Ville`, `Temperature`, `Description`, `Humidite`, `DateHeure`
   - `ResultatMeteo`: propriétés `Meteo`, `Succes`, `MessageErreur`
   - `CacheEntry`: propriétés `Ville`, `Meteo`, `DateExpiration`

   **Services/**
   - `MeteoService`:
     - Champ `private readonly HttpClient _client`
     - Champ `private readonly string _apiKey`
     - Méthode `public async Task<ResultatMeteo> ObtenirMeteoAsync(string ville, CancellationToken token)`
     - Méthode `public async Task<List<ResultatMeteo>> ObtenirPlusieursVillesAsync(List<string> villes, IProgress<int> p`
   - `CacheService`:
     - Champ `private Dictionary<string, CacheEntry> _cache`
     - Méthode `public bool TryGetMeteo(string ville, out Meteo meteo)`
     - Méthode `public void AjouterCache(string ville, Meteo meteo)`
   - `FichierService`:
     - Méthode `public async Task SauvegarderCsvAsync(List<Meteo> donnees, string chemin)`

   **Views/**
   - `MeteoView`:
     - Méthode `public void AfficherTableauMeteo(List<ResultatMeteo> resultats)`
     - Méthode `public void AfficherProgression(int pourcentage, string ville)`
     - Méthode `public void AfficherErreur(string message)`
2. Utilisez une API météo gratuite (OpenWeatherMap, WeatherAPI)
3. Villes suggérées: Paris, Londres, New York, Tokyo, Sydney
4. Gérez toutes les erreurs et l'annulation
5. Implémentez le cache (durée: 10 minutes)

**Architecture attendue:**

```
Projet/
  Models/
    Meteo.cs
    ResultatMeteo.cs
    CacheEntry.cs
  Services/
    MeteoService.cs
    CacheService.cs
    FichierService.cs
  Views/
    MeteoView.cs
  Program.cs
```

**Villes suggérées:** Paris, Londres, New York, Tokyo, Sydney

**Bonus:** Ajoutez un système de cache pour éviter de télécharger les mêmes données plusieurs fois en moins de 10 minutes.