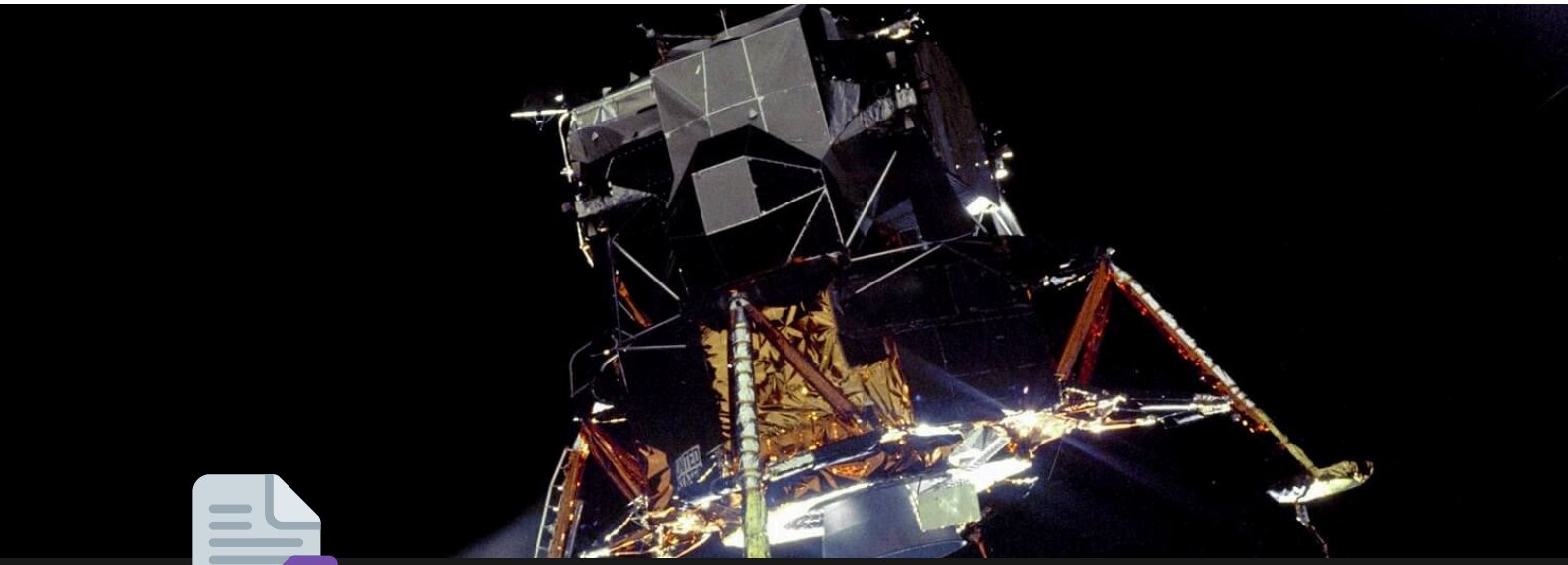

Prosit Retour 4

Infos et rôles :

- **Date** : 10/02/2025
- **Heure de début : 15h05**
- **Heure de fin : 15h50**
- **Durée : 45Min**
- **Promo :** A3 **FISA INFO**

- **Tuteur** : Mohammed
- **Animateur** : Hugo
- **Secrétaire** :
- **Scribe** : Vincent
- **Gestionnaire** : Erwan

Mots-clés :

- **Maquette logiciel :**

  Représentation visuelle ou fonctionnelle simplifiée d'un logiciel pour en valider l'interface ou les fonctionnalités avant son développement complet.

- **Composant COM :**

  Technologie Microsoft permettant la création de composants logiciels indépendants et réutilisables, facilitant la communication entre différentes applications.

- **MMU (Memory Management Unit) :**

  Unité matérielle intégrée au processeur qui gère la mémoire virtuelle en traduisant les adresses logiques des programmes en adresses physiques.

- **Composant métier :**

  Module logiciel contenant la logique et les règles de gestion propres à une activité spécifique, comme la gestion des patients ou des commandes.

- **Bus de traitement :**

  Mécanisme de communication permettant le transfert et l'orchestration des données entre différents modules ou composants logiciels.

- **Crash mémoire :**

  Défaillance du système ou du programme provoquée par un accès erroné à la mémoire, souvent dû à un dépassement de capacité ou un pointeur incorrect.

- **Unité de traitement :**

  Composant matériel (comme le CPU ou GPU) responsable de l'exécution des instructions logiques, arithmétiques et des opérations de contrôle.

- **Chaîne de valeur :**

  Ensemble des étapes où des processus et des ressources sont combinés pour créer de la valeur ajoutée à un produit ou service final.

- **Simulation :**

  Processus consistant à reproduire le fonctionnement d'un système réel via un modèle numérique ou virtuel afin d'en tester les comportements.

Solutions :

Louis / Martin :

différente memoire :

- registre : ultra rapide dans le processeur,
- cache : memoire intermédiaire entre registre
- logique : memoire physique utilisé par les programmes
- physique : memoire réelle dans un ordi comme la ram / plus rapide que SSD

Simon :
stack :

Heap : zone memoire pour stockage dynamique. classe existante comme free

Tableau comparaison de Simon :

ère	Pile (Stack)	Tas (Heap)
sse	Très rapide	Plus lent
sation	Variables locales, appels de fonction	Objets (instances de classes), données dynamiques
moire libérée	Automatiquement (fin de fonction)	Par le Garbage Collector
cture	LIFO (dernier entré, premier sorti)	Pas de structure spécifique
e	Limitée	Beaucoup plus grande

Mathéo :

- Optimal : remplacer la page pas utilisé dans le futur (dur)
- fifo : on remplace la plus ancienne page
- NRU : Remplace celle pas utilisé récemment

Martin :

- pagination : division de la memoire en bloc de taille fixe
- segmentation : decoupage en section logique

Tom :

- LRU : remplace la page remplacer depuis le plus longtemps
- LFU : remplace la page la moins fréquemment utilisé

Martin W :
Allocation fragmenté ou contigu : Beaucoup de mot très rapide

Martin :
Garbage collector :

- Permet d’automatiser la liberation de l’espace :
  - marquage
  - liberation
  - ?
  - Exemple de classe fais maison

```
class Ressource : IDisposable { private bool disposed = false; public void Dispose() { if (!disposed) { // Libérer les ressources disposed = true; } } }
```

- Idisposable :
  - tableau de simon :

nparaison entre GC et IDisposable:

ère	IDisposable	Garbage Collector (GC)
ctif	Libérer **manuellement** des ressources non managées (fichiers, connexions, etc.)	Gérer **automatiquement** la mémoire des objets managé
e de mémoire concernée	Ressources **non managées** (fichiers, sockets, base de données, etc.)	Objets **managés** (créés en C avec `new`)
node principale	`Dispose()`	`GC.Collect()` (géré automatiquement)
nd intervient-il ?	Dès que l'on appelle `Dispose()` (ou avec `using`)	Automatiquement, lorsque la mémoire devient insuffisante
trôle par le développeur ?	**Oui**, il faut appeler `Dispose()`	**Non**, c'est le .NET Runtime q décide quand collecter
act sur la performance	**Très rapide**, car immédiat	**Plus lent**, car il doit analyser mémoire et nettoyer
ion des objets	S'applique aux objets contenant des ressources externes	Gère uniquement les objets alloués dans le tas (heap)
sation typique	`FileStream`, `SqlConnection`, `Socket`, `Bitmap`	Tous les objets créés avec `n`
oin d'un finaliseur `.ass()`) ?	**Parfois**, si on veut garantir la libération même sans `Dispose()`	**Non nécessaire**, car le GC s occupe

- delegation : Thomas
  - Plus dynamique que des classes structurel
  - tres flexible mais peu lisible
  - classe intermediaire plus la pour methode amener a evoluer

- Lambda function : Erwan
  - Methode sans nom pour des calculs simples
  - qui ne va pas etre reutiliser donc ne pas les declarer
  - plusieurs type :
    - lambda Action : retourne rien
    - lambda Function : retourne quelque chose comme un integer

MMU : Erwan / Louis

```
using System; using System.Collections.Generic; namespace Workshop4 {
class Program { static void Main(string[] args) { // Demander à
l'utilisateur combien d'unités de traitement il souhaite utiliser
Console.WriteLine("Combien d'unités de traitement voulez-vous utiliser
?"); int nombreUnites = Convert.ToInt32(Console.ReadLine()); // Liste pour
stocker l'ordre d'exécution des unités List<int> ordreExecution = new
List<int>(); // Demander à l'utilisateur l'ordre des unités de traitement
Console.WriteLine("Veuillez entrer l'ordre des unités de traitement :");
string ordreInput = Console.ReadLine(); // Traitement de l'entrée de
l'utilisateur pour extraire l'ordre des unités string[] ordreArray =
ordreInput.Split(","); foreach (var unite in ordreArray) { // Extraire le
numéro de chaque unité et l'ajouter à la liste if (unite.StartsWith("U"))
{ if (int.TryParse(unite.Substring(1), out int numUnite)) {
ordreExecution.Add(numUnite); } else { Console.WriteLine($"Erreur : Unité
invalide {unite}"); } } } // Simulation de l'exécution des unités de
traitement Console.WriteLine("\nExécution des unités de traitement :");
foreach (int unite in ordreExecution) { // Ici, on simule le traitement de
chaque unité Console.WriteLine($"Unité U{unite} en cours de
traitement..."); } // Fin de la simulation Console.WriteLine("\nSimulation
terminée."); } } }
```