**Description du scénario**

Imaginez que nous avons un système de gestion de fichiers qui permet aux utilisateurs de charger et télécharger des fichiers. Nous voulons restreindre l'accès aux utilisateurs en fonction de leurs droits. Seuls les utilisateurs qui ont les droits appropriés peuvent accéder à certains fichiers. Pour ce faire, nous allons utiliser le design pattern Proxy.

**Diagramme UML**

Voici une représentation simplifiée de la structure de notre système:

scss

+---------------------+

| IFileAccess |

+---------------------+

| + LoadFile() |

| + DownloadFile() |

+----------+----------+

^

|

+--------------+--------------+

| |

+-------------v-----------+ +-------------v-----------+

| RealFileAccess | | FileAccessProxy |

+-------------------------+ +-------------------------+

| + LoadFile() | | + LoadFile() |

| + DownloadFile() | | + DownloadFile() |

+-------------------------+ +-------------------------+

Dans ce diagramme:

* IFileAccess est une interface définissant les opérations de base qui peuvent être effectuées sur un fichier, telles que LoadFile et DownloadFile.
* RealFileAccess est la classe concrète qui implémente l'interface IFileAccess et fournit la fonctionnalité réelle de chargement et de téléchargement des fichiers.
* FileAccessProxy est la classe Proxy qui implémente également l'interface IFileAccess. Elle contrôle l'accès aux opérations de RealFileAccess.

**Implémentation en C#**

Maintenant, implémentons ce scénario en C# en utilisant le design pattern Proxy.

csharp

// Interface defining the operations that can be performed on a file

public interface IFileAccess {

void LoadFile(string fileName);

void DownloadFile(string fileName);

}

// The RealSubject class

public class RealFileAccess : IFileAccess {

public void LoadFile(string fileName) {

//...Load file

Console.WriteLine($"Loading file: {fileName}");

}

public void DownloadFile(string fileName) {

//...Download file

Console.WriteLine($"Downloading file: {fileName}");

}

}

// The Proxy class

public class FileAccessProxy : IFileAccess {

private RealFileAccess realFileAccess;

private string currentUserRole;

public FileAccessProxy(string currentUserRole) {

this.currentUserRole = currentUserRole;

this.realFileAccess = new RealFileAccess();

}

public void LoadFile(string fileName) {

// Check user role before access

if (currentUserRole == "Admin" || currentUserRole == "Manager") {

realFileAccess.LoadFile(fileName);

} else {

Console.WriteLine("Access Denied: Insufficient privileges");

}

}

public void DownloadFile(string fileName) {

// Check user role before access

if (currentUserRole == "Admin") {

realFileAccess.DownloadFile(fileName);

} else {

Console.WriteLine("Access Denied: Insufficient privileges");

}

}

}

**Utilisation de l'implémentation**

csharp

// Create a proxy for a user with role "User"

IFileAccess userProxy = new FileAccessProxy("User");

userProxy.LoadFile("document.txt"); // Access Denied

userProxy.DownloadFile("document.txt"); // Access Denied

// Create a proxy for a user with role "Admin"

IFileAccess adminProxy = new FileAccessProxy("Admin");

adminProxy.LoadFile("document.txt"); // Loading file: document.txt

adminProxy.DownloadFile("document.txt"); // Downloading file: document.txt

Dans cet exemple, nous avons utilisé le pattern Proxy pour contrôler l'accès aux opérations de chargement et de téléchargement de fichiers en fonction du rôle de l'utilisateur.