



Traitements des exceptions  
et exceptions utilisateur

# Objectifs

- Exception
- Traitement : try / catch / throw / finally
- Exception utilisateur

# Exception 1 / 2

- Un exception représente une erreur ou un comportement non attendu durant l'exécution d'un programme
- Causes :
  - Erreur dans un appel (précondition non respectée)
  - Erreur dans le code (bug)
  - Erreur de ressources (mémoire, fichier, sécurité, etc.)
- Exemples :
  - Une fonction d'écriture dans un fichier, mais le disque est plein
  - Une fonction pour envoyer un paquet réseau, mais la connexion est coupée durant l'envoie
  - Une fonction pour lire un « double » au clavier, mais l'utilisateur entre autre chose que des chiffres

# Exception 2 / 2

- Vous devez réagir une erreur (dans le code) :
  - Journaliser l'erreur
  - Retourner à un comportement normal
  - Libérer des ressources (mémoire, fichier, connexion réseau, etc.)
- Pourquoi journaliser ?
  - Intention d'analyser le journal
  - Possibilité d'améliorer le système
    - Qualité du code ?
    - Utilisation non prévue ?
    - Problème de communications inter-systèmes ?
    - Problème d'infrastructure ?
  - Sécurité

# Types d'exceptions – C#

Type Exception	Description	Example
<a href="#">Exception</a>	Classe mère de toutes les exceptions	Aucun (utilisez des classes filles de cette exception)
<a href="#">IndexOutOfRangeException</a>	Lancée par <b>le système</b> quand une collection est accédée à un indice en dehors des limites	tbl[ <b>tbl.Length</b> ]
<a href="#">NullReferenceException</a>	Lancée par <b>le système</b> quand une valeur	object o = null; <b>o.ToString();</b>
<a href="#">InvalidOperationException</a>	Lancée par <b>une méthode</b> quand elle n'est pas applicable	Appeler Enumerator.MoveNext() après avoir supprimé un élément de la collection correspondante
<a href="#">ArgumentException</a>	Classe mère des exception touchant les paramètres	Aucun (utilisez une classe qui hérite d'elle)
<a href="#">ArgumentNullException</a>	Lancée par <b>une méthode</b> quand elle n'accepte pas une valeur nulle en paramètres	String s = null; "Calculate".IndexOf( <b>s</b> );
<a href="#">ArgumentOutOfRangeException</a>	Lancée par <b>une méthode</b> qui valide le domaine de valeurs de ses paramètres	String s = "string"; s.Substring( <b>s.Length+1</b> );

# Lever une erreur

- Utilisez le mot « `throw` » suivi de **l'objet** à lancer

```
throw new ArgumentException(nameof(p_indiceASupprimer));
```

# Traitement des exceptions – C# – Structure « try »

- Bloc « try » : bloc où l'on peut capturer une erreur
- Blocs « catch » : blocs qui sont passés en revue afin de trouver le bloc correspondant à l'erreur
  - Un seul bloc est exécuté
  - La recherche s'effectue de haut en bas :
    - Le filtre des blocs catch les plus hauts doivent être les plus spécifiques
    - Plus on bas, plus on est générale
- Bloc « finally » : bloc exécuté qu'il y ait une erreur ou non

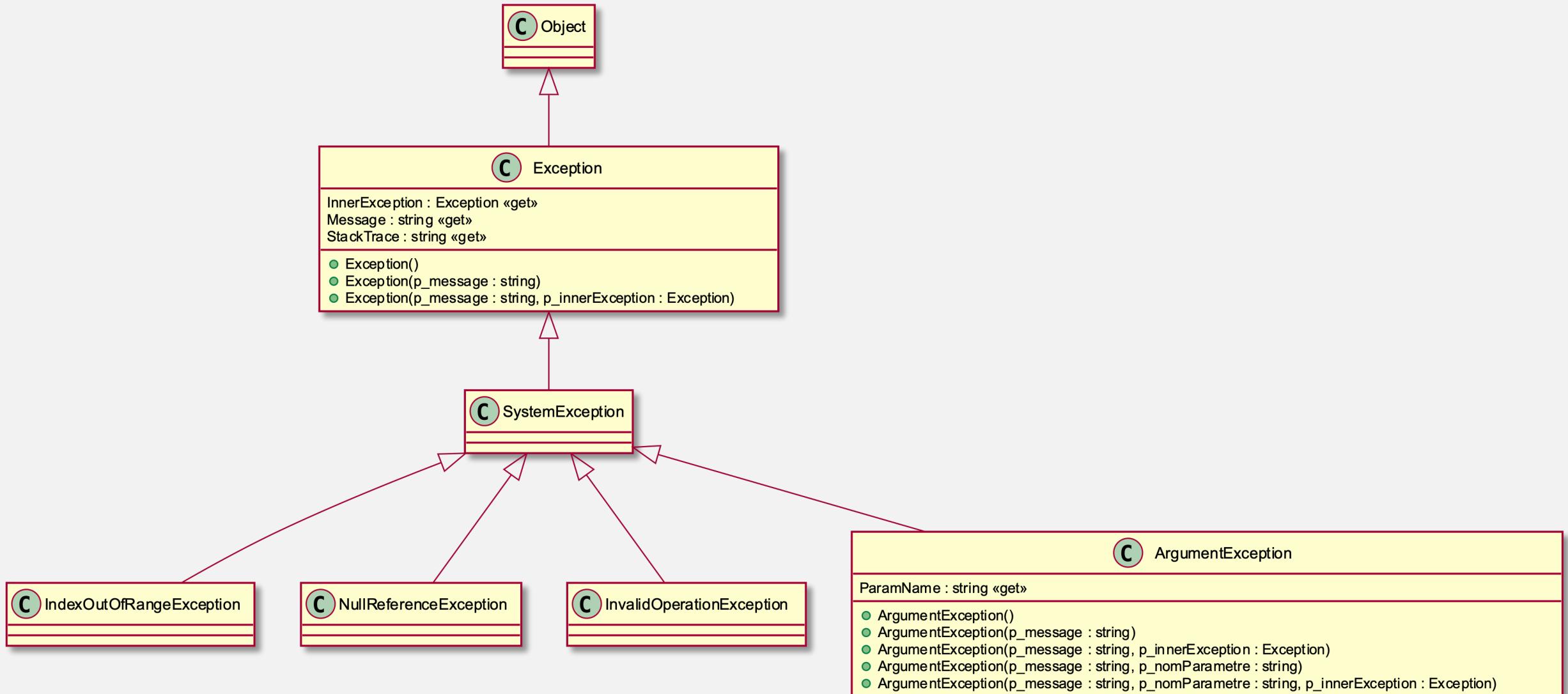
# Traitemen~~t~~t des exceptions – C#

```
try
{
    // Bloc de code qui peut lever une exception
    using (StreamReader sr = new StreamReader("Un nom de fichier.txt", Encoding.UTF8));
    {
        // ...
    }
}
catch (FileNotFoundException ex)
{
    // Fichier non trouvé
}

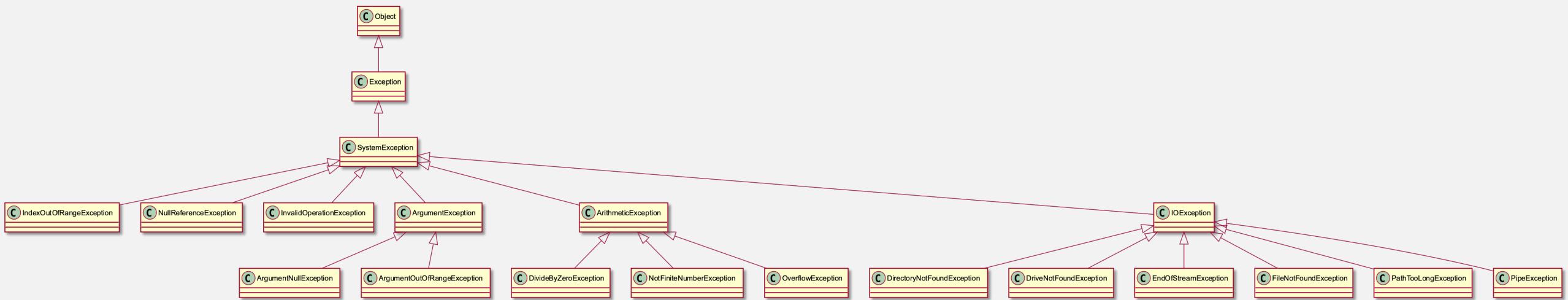
catch (IOException ex)
{
    // Autres erreurs ?

    throw; // Relancer la même exception sans perdre la StackTrace
}
finally
{
    // Libération de ressources : fermeture de fichiers et de connexions, etc.
}
```

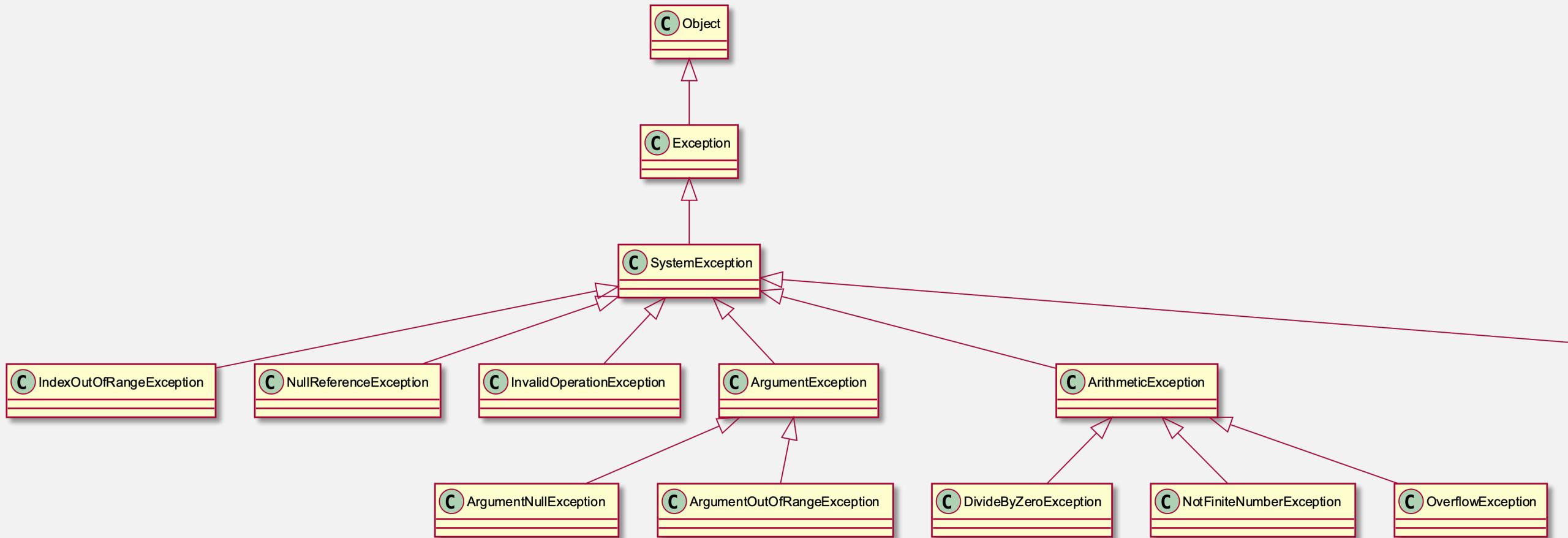
# Membres de quelques Exceptions .Net



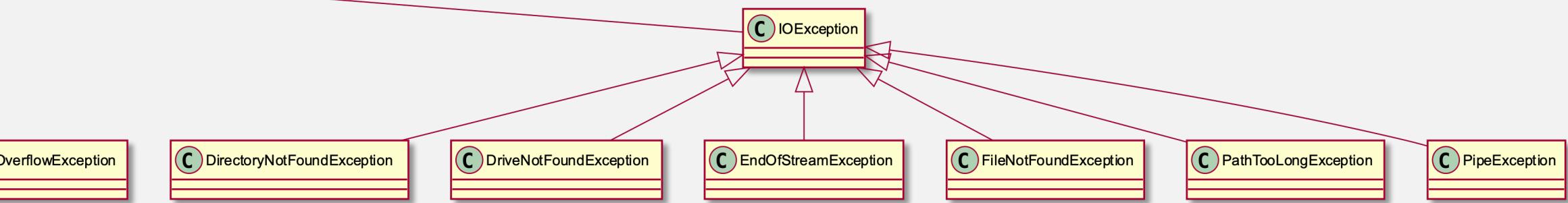
# Quelques exceptions de .Net



# Quelques exceptions de .Net



# Quelques exceptions de .Net



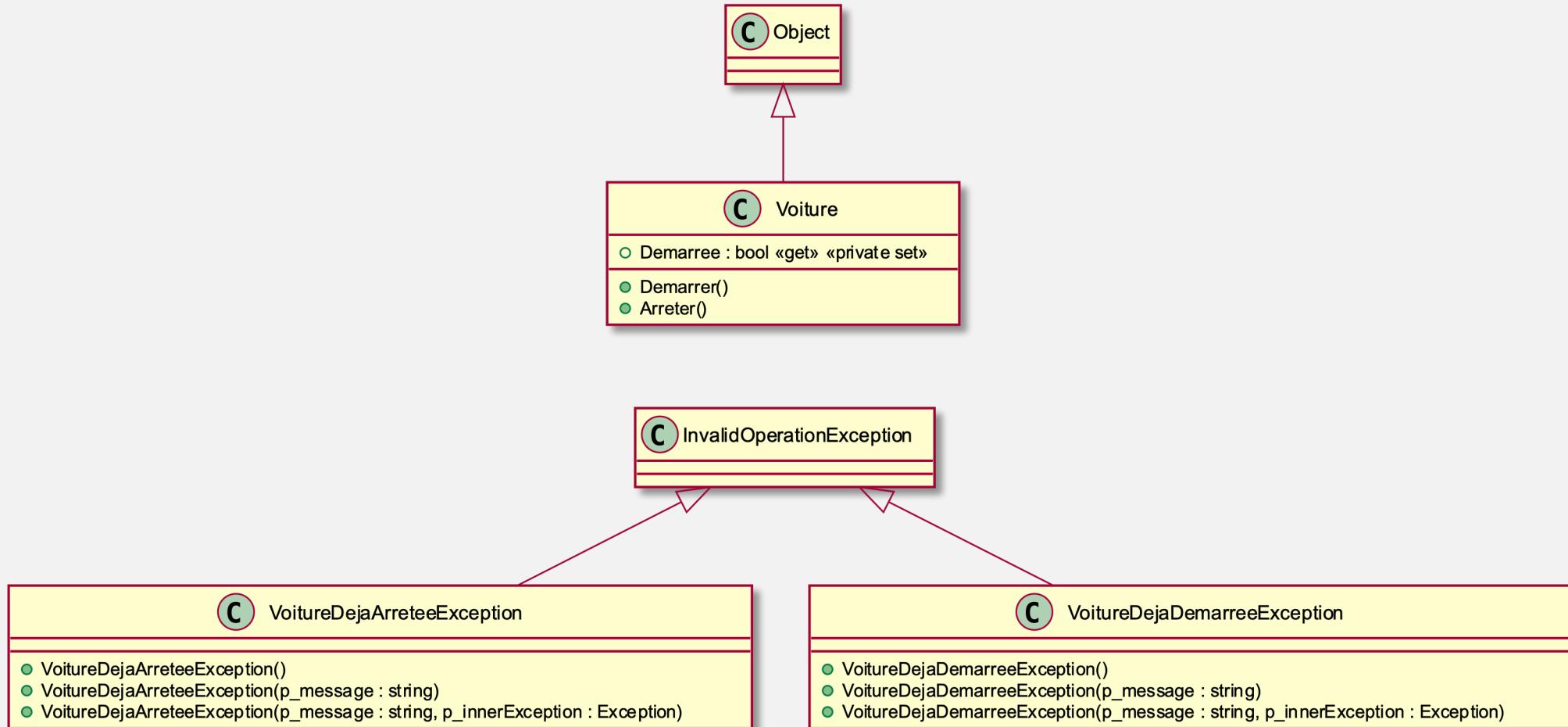
# Que tirer du diagramme précédent ?

- Le **nom** de la classe de l'exception **doit indiquer l'erreur**
  - Des propriétés générales : *Message*, *StackTrace*, etc.
  - Des propriétés spécifiques à un type d'erreur : *ParamName*, etc.
- Les **exceptions** sont généralement **organisées** en une **hiérarchie**
- Il y a beaucoup de types d'exceptions dans un cadriel : vous devriez trouver votre bonheur dans les types d'erreurs
- Les exceptions sont implantées avec des classes avec « **Exception** » comme **classe de base**  
=> Pour définir vos exceptions vous allez **hériter** de la classe « **Exception** » ou **d'une de ses classes filles**

# Exception utilisateur

- Vous devez minimalement hériter de la classe « *Exception* »
- Vous devriez aussi essayer de trouver des classes filles plus appropriées. Exemple : *ArgumentException*, *InvalidOperationException*, etc.
- Définissez les constructeurs suivants :
  - ctor()
  - ctor(p\_message : string)
  - ctor(p\_message : string, p\_innerException: Exception)

# Exemple – Classe Voiture



# Exemple – Classe Voiture

```
public class Voiture
{
    public bool Demarree { get; private set; }

    public void Demarrer()
    {
        if (this.Demarree)
        {
            throw new VoitureDejaDemarreeException();
        }

        this.Demarree = true;
    }

    public void Arreter()
    {
        if (!this.Demarree)
        {
            throw new VoitureDejaArreteException();
        }

        this.Demarree = false;
    }
}
```

# Exemple – Classes d’erreurs

```
public class VoitureDejaDemarreeException : InvalidOperationException
{
    public VoitureDejaDemarreeException()
    { ; }

    public VoitureDejaDemarreeException(string message) : base(message)
    { ; }

    public VoitureDejaDemarreeException(string message, Exception innerException) :
    base(message, innerException)
    { ; }
}
```

# Exemple – Classes d’erreurs

```
public class VoitureDejaArreteException : InvalidOperationException
{
    public VoitureDejaArreteException()
    { ; }

    public VoitureDejaArreteException(string message) : base(message)
    { ; }

    public VoitureDejaArreteException(string message, Exception innerException) :
base(message, innerException)
    { ; }
}
```

# Exemple – Un programme

```
public static void DemoVoiture()
{
    int choixMenu = 0;
    Voiture voiture = new Voiture();
    do
    {
        Console.Out.WriteLine("1. Démarrer");
        Console.Out.WriteLine("2. Arrêter");
        Console.Out.WriteLine("9. Quitter");

        choixMenu = Console.In.ReadInt();

        try
        {
            switch (choixMenu)
            {
                case 1:
                    voiture.Demarrer();
                    break;
                case 2:
                    voiture.Arreter();
                    break;
                case 9:
                    break;
                default:
                    throw new InvalidOperationException("Option incorrecte");
            }
        }
    }
```

```
        } while (choixMenu != 9);

        catch (VoitureDejaDemarreeException ex)
        {
            Console.Error.WriteLine("La voiture est déjà démarrée");
        }
        catch (VoitureDejaArreteeeException ex)
        {
            Console.Error.WriteLine("La voiture est déjà arrêtée");
        }
        catch (InvalidOperationException ex)
        {
            Console.Error.WriteLine($"Opération invalide : {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.Error.WriteLine($"Erreur non prévue : {ex.Message}");
        }
    }
}
```

*Ici, c'est un exemple sur les exceptions et non sur ce qu'il faut faire : vous devez valider les entrées utilisateur et éviter les exceptions*

# Références

- Bonnes pratiques par Microsoft : <https://docs.microsoft.com/en-us/dotnet/standard/exceptions/>