

# Diagramme de classes

## Raisonnement et abstraction (2)

# Modélisation arborescente

- Module : Algo. Avancée → Classe Arbre

Algorithmique avancée  
Florent Becker


Arbres  
Vocabulaire et algorithmes de base  
Implémentation de la classe Arbre  
Parcours en profondeur  
Chemins  
Parcours en largeur

XML  
Syntaxe

## Attributs

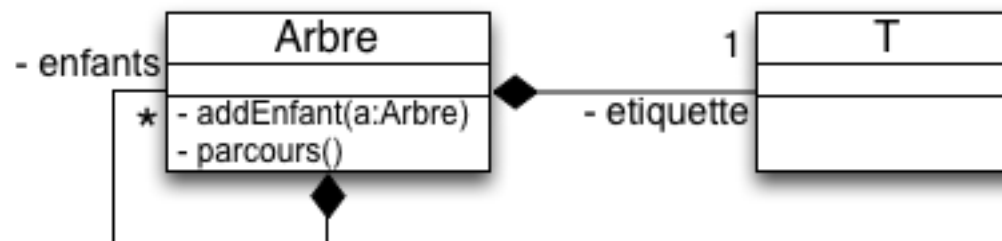
Quels sont les attributs nécessaires pour la classe `Arbre<T>` ?

- Une étiquette : `T etiquette`
- Une liste d'enfants : `ArrayList<Arbre<T> > enfants;`

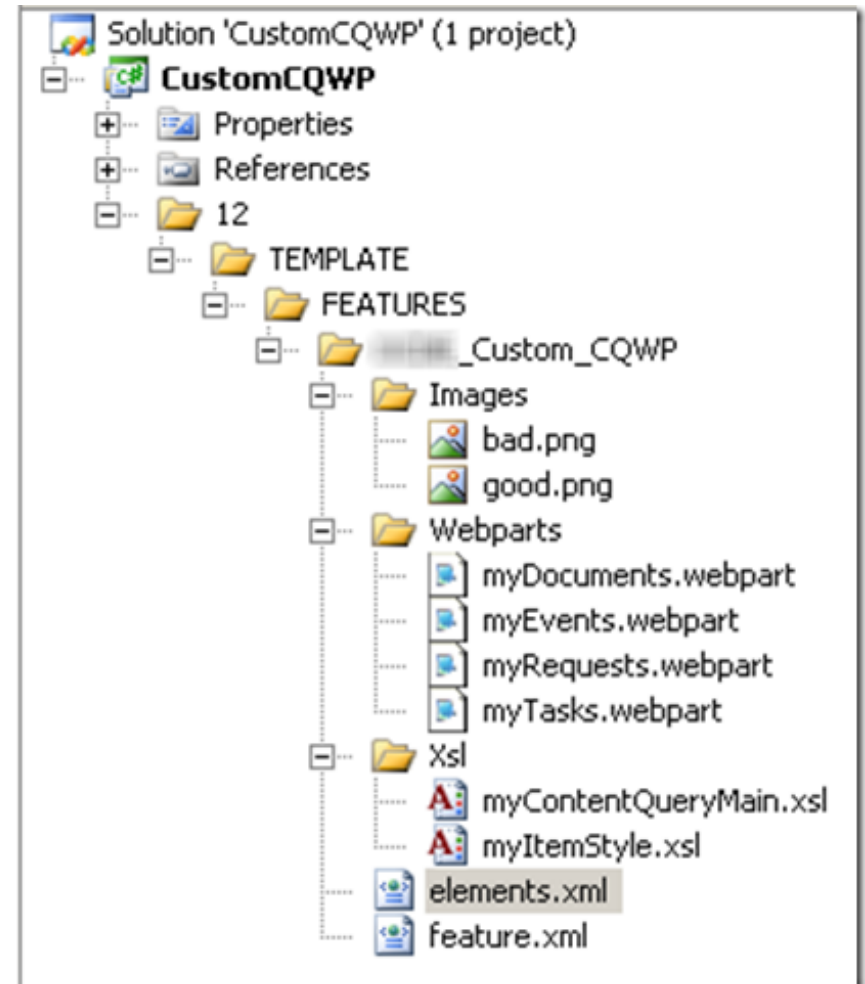
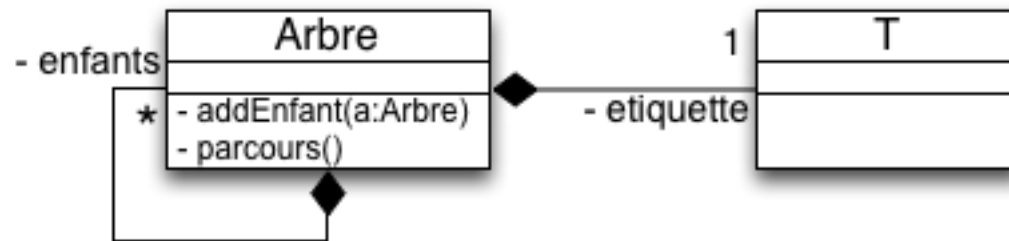


# Modélisation arborescente

- Module : Algo. Avancée  $\rightarrow$  Classe Arbre



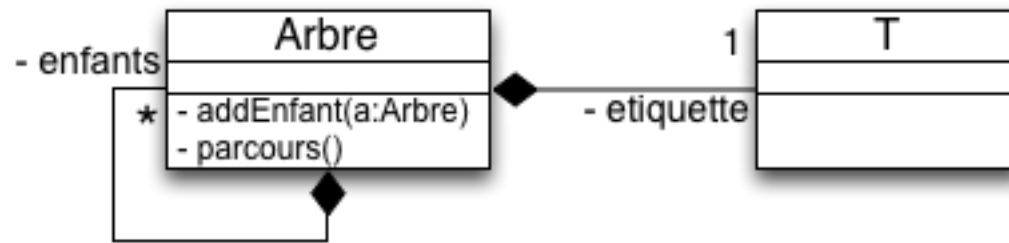
# Modélisation arborescente



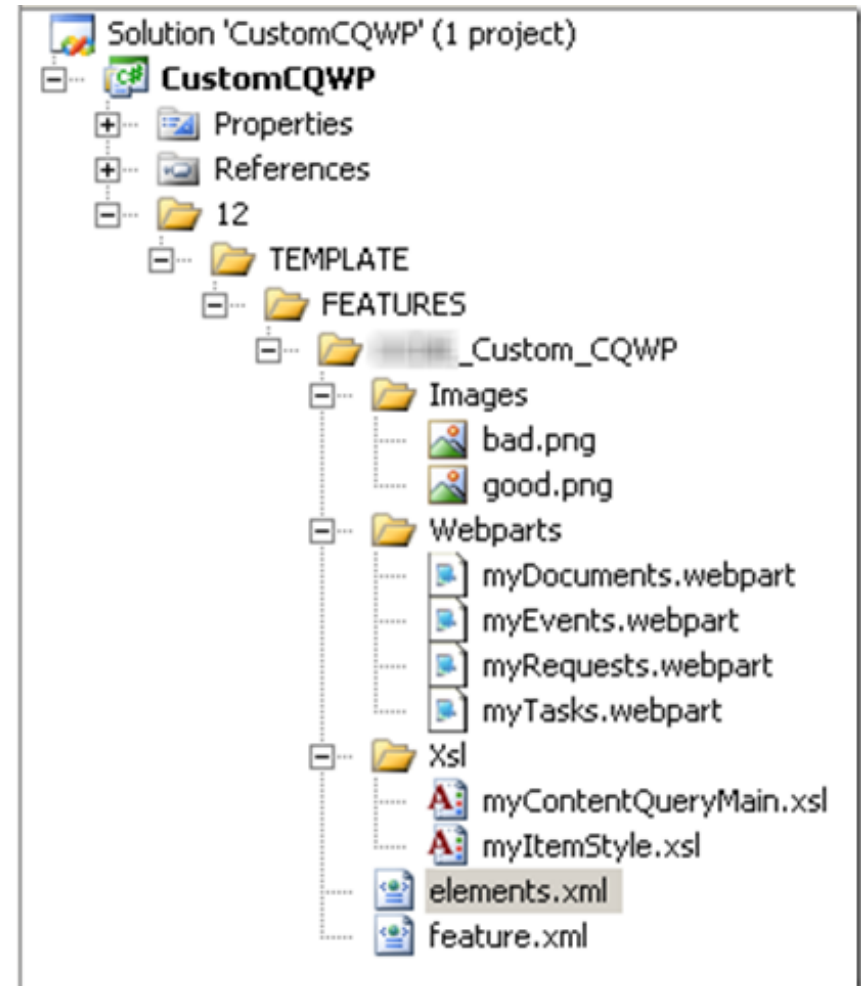
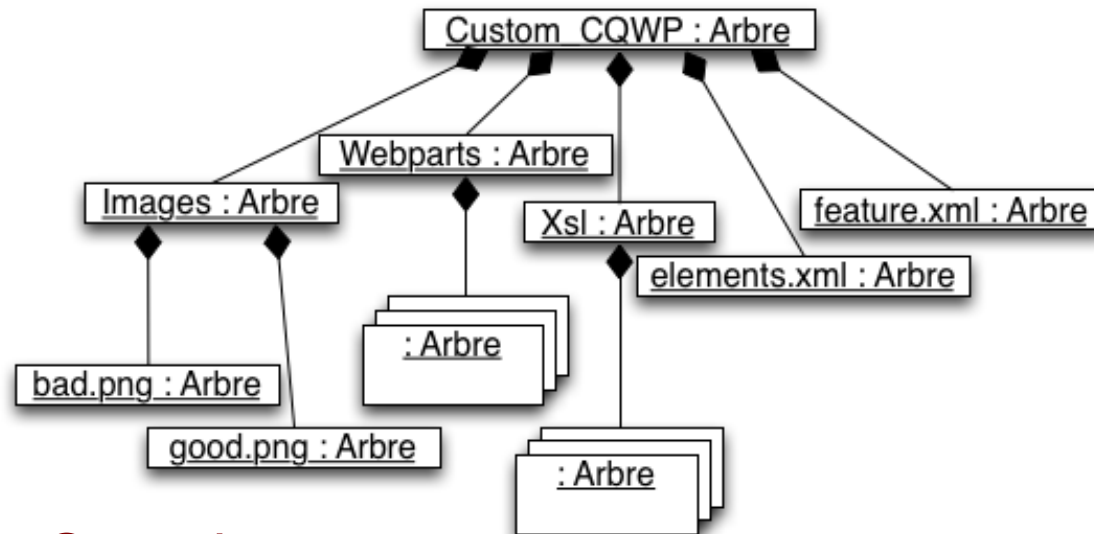
## Question

Cette modélisation est-elle adaptée pour implémenter une arborescence de fichiers ?

# Modélisation arborescente



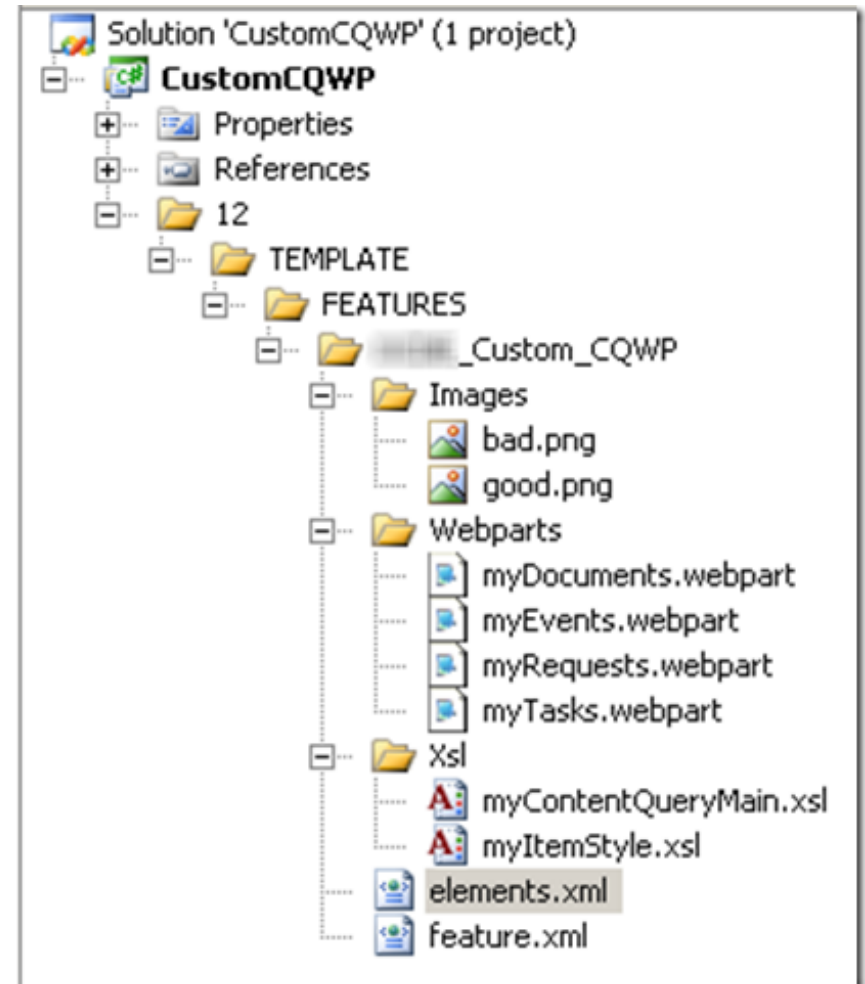
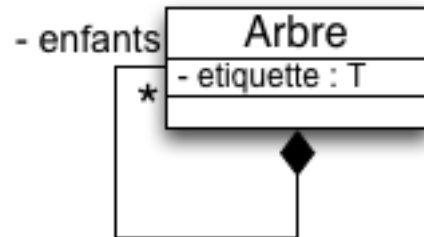
## Diagramme d'objets :



## Question

Cette modélisation est-elle adaptée pour implémenter une arborescence de fichiers ? **A PRIORI OUI**

# Modélisation arborescente

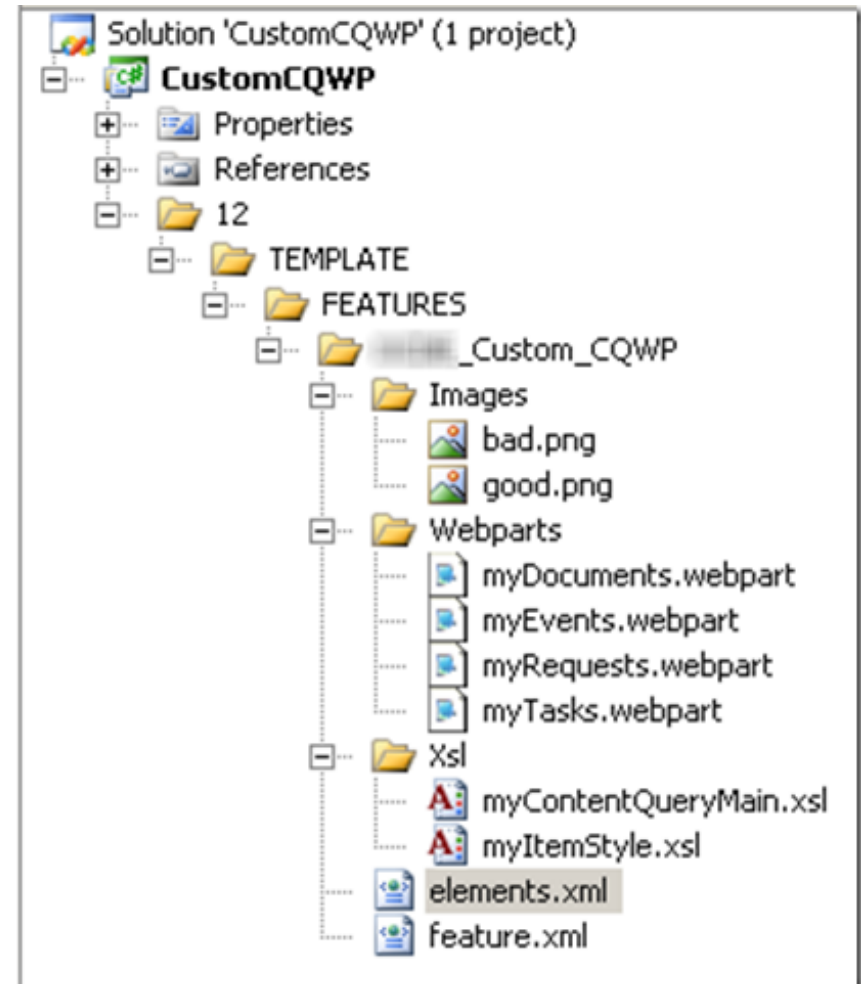
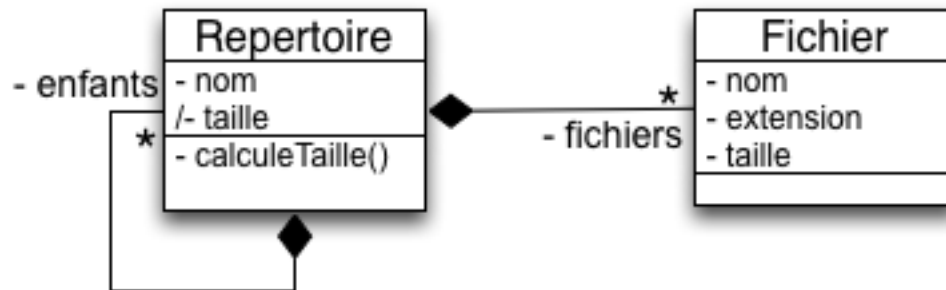


A modéliser

*A chaque fichier est associé : une extension, une taille, etc.*

*Un répertoire n'a pas d'extension et sa taille peut être calculée.*

# Modélisation arborescente

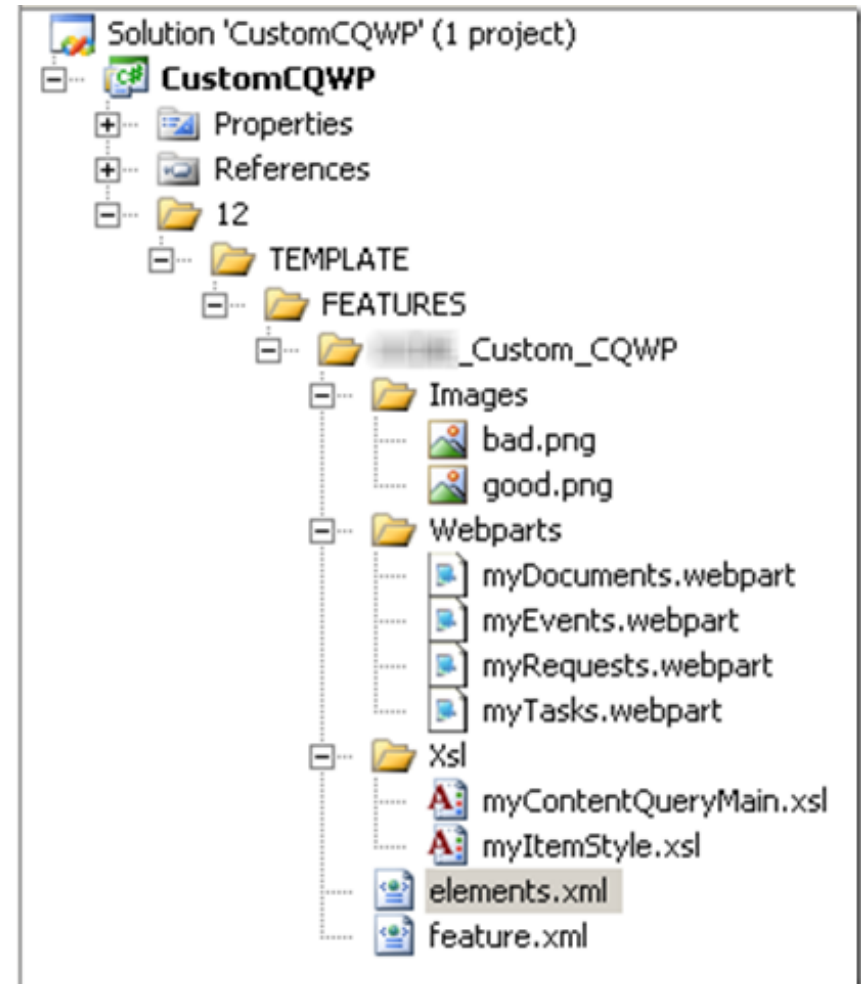
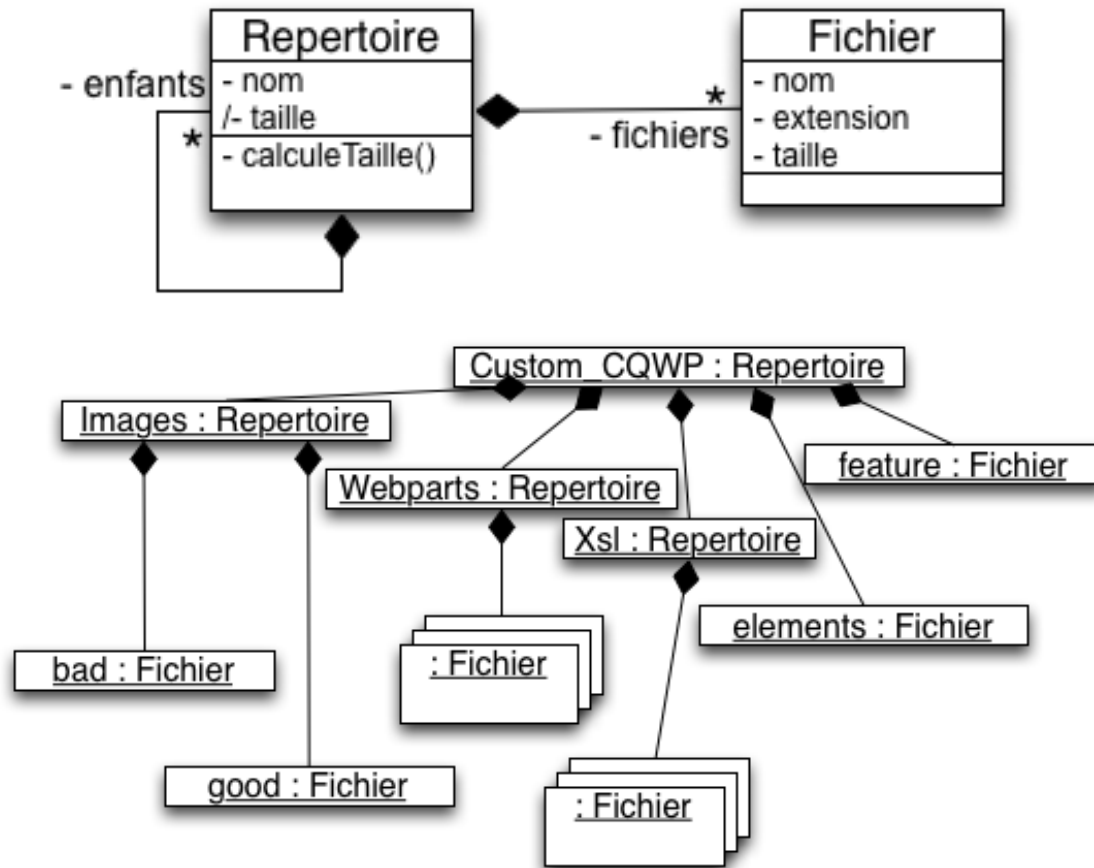


A modéliser

*A chaque fichier est associé : une extension, une taille, etc.*

*Un répertoire n'a pas d'extension et sa taille peut être calculée.*

# Modélisation arborescente



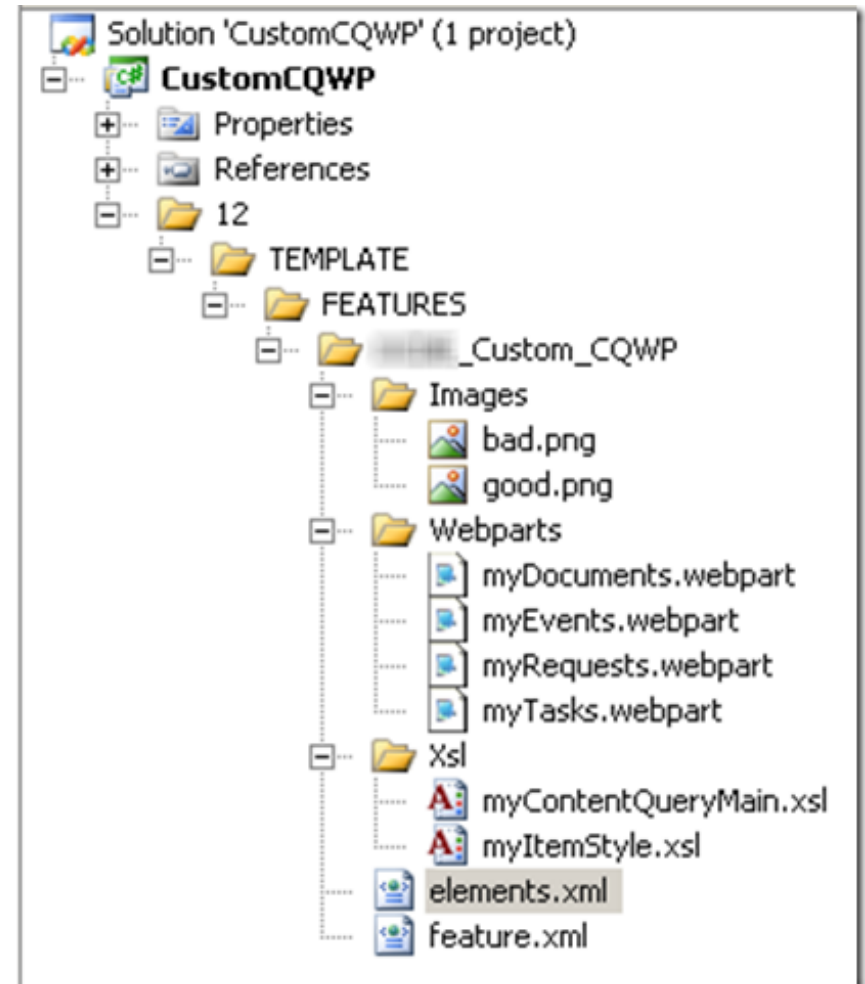
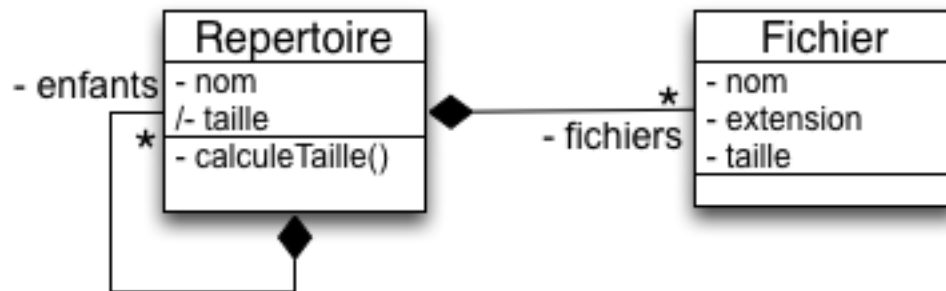
A modéliser

*A chaque fichier est associé : une extension, une taille, etc.*

*Un répertoire n'a pas d'extension et sa taille peut être calculée.*



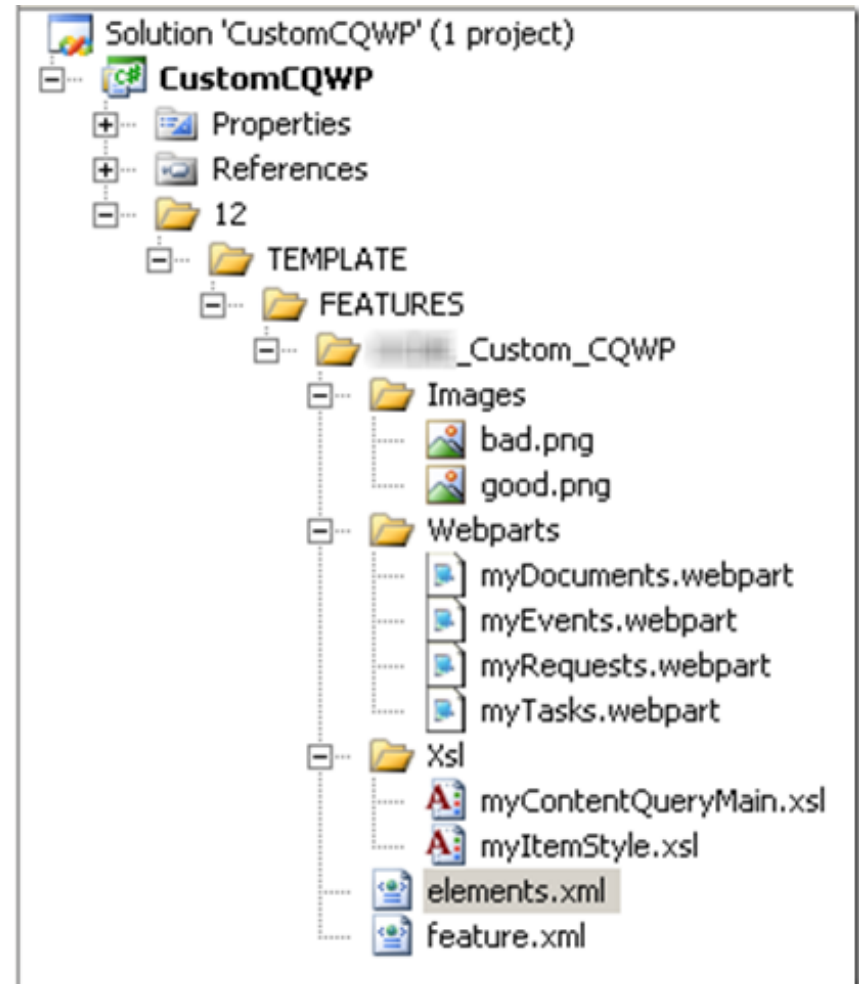
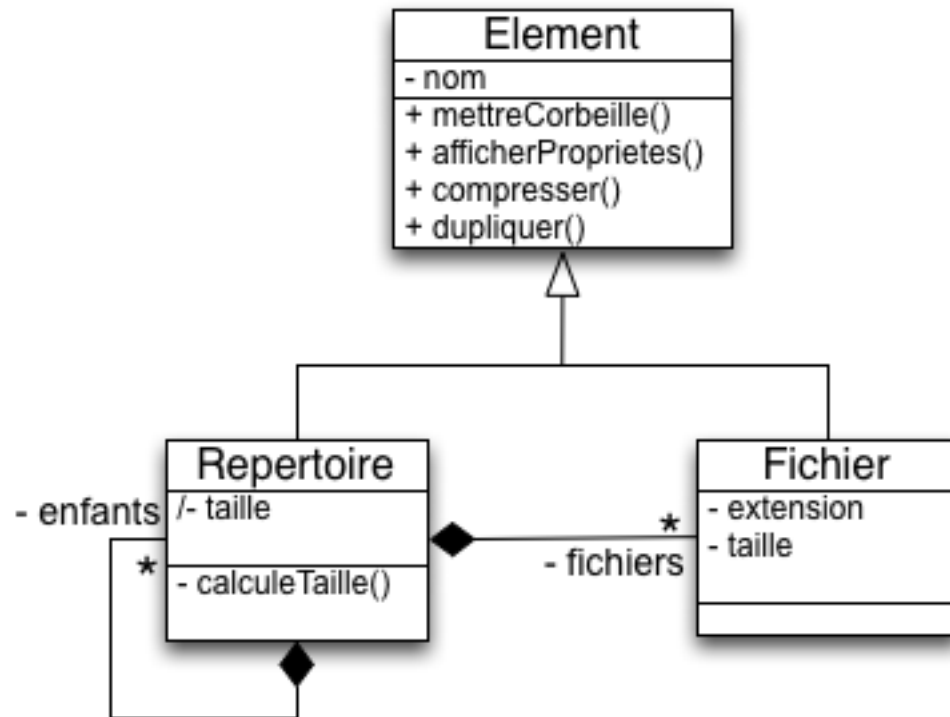
# Modélisation arborescente



A modéliser

*Il y a des comportements communs aux Répertoires et aux fichiers : afficher les propriétés, mettre à la corbeille, etc.*

# Modélisation arborescente



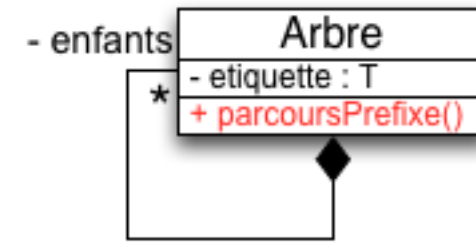
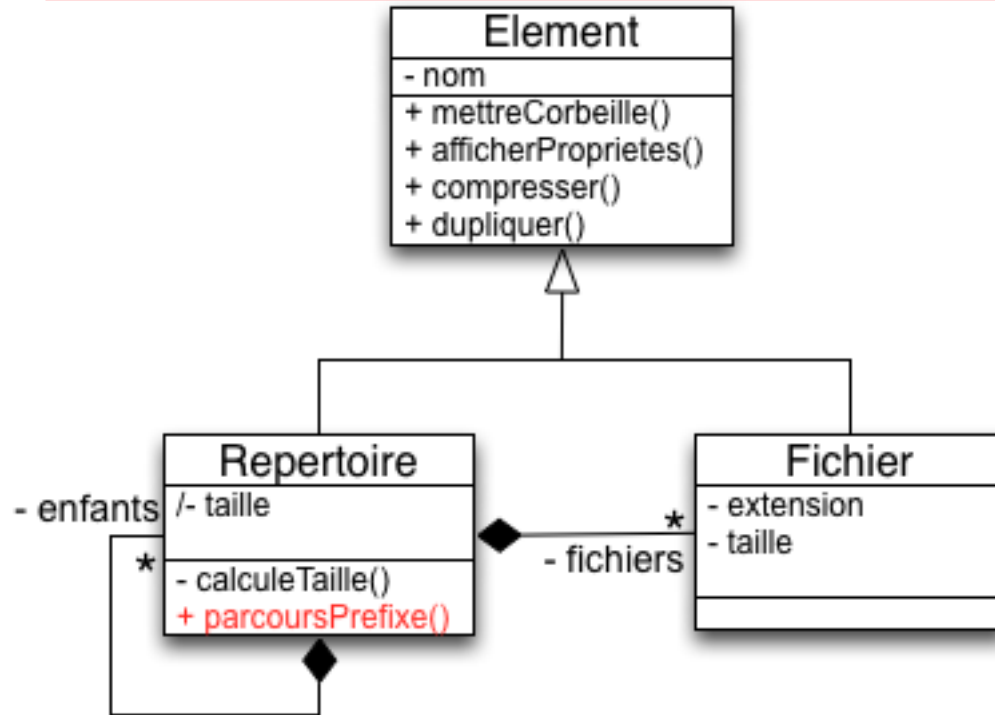
A modéliser

*Il y a des comportements communs aux Répertoires et aux fichiers : afficher les propriétés, mettre à la corbeille, etc.*

## Question

Quid des processus récur­sifs (ex. parcours) sur les arbres ?

→ traitements dissociés : :Repertoire vs. :Fichier !!



```

Void parcoursPrefixe(){
    traitement() ; //à remplacer (ex. affiche le nom)
    for(Repertoire r : enfants)
        r.parcoursPrefixe() ;
    for(Fichier f : fichiers)
        f.traitement();
}
    
```

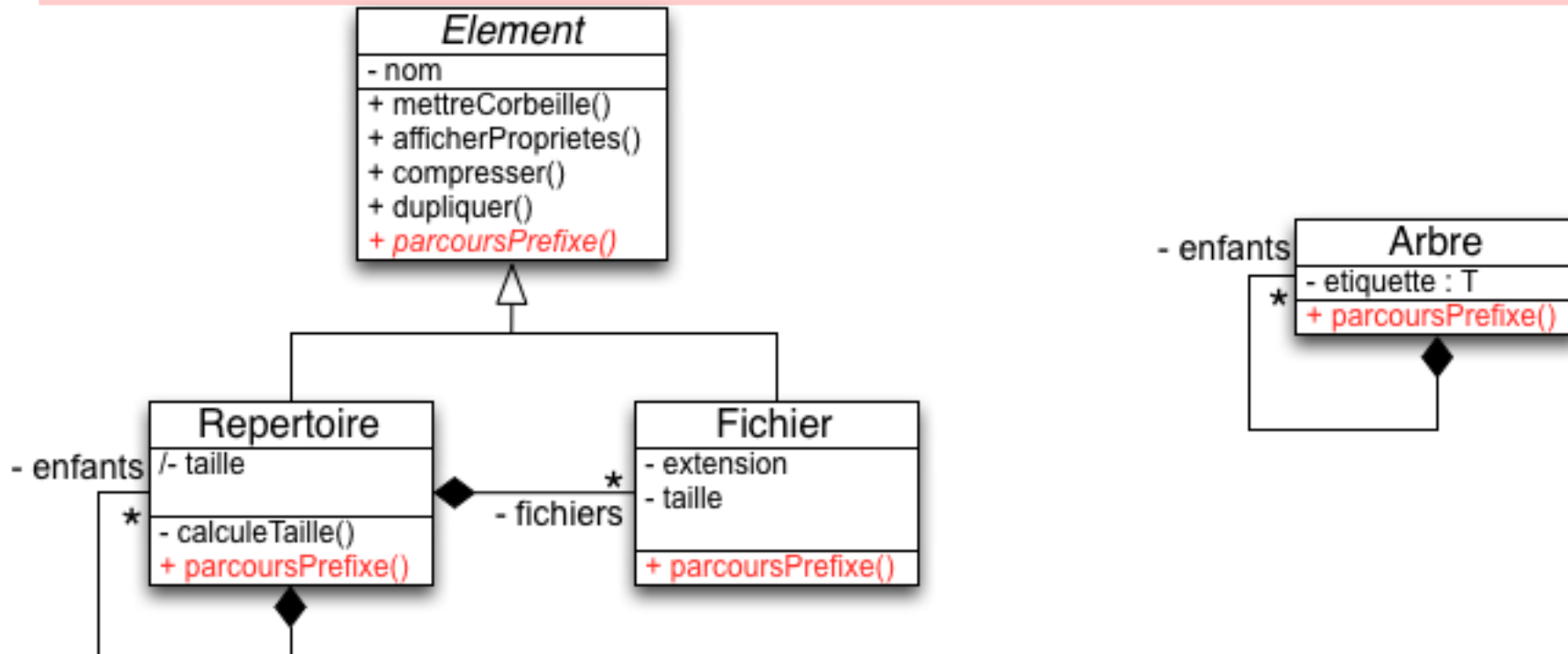
```

Void parcoursPrefixe(){
    traitement() ; //à remplacer
    for(Arbre<T> e : enfants)
        e.parcoursPrefixe() ;
}
    
```

## Question

Quid des processus récurifs (ex. parcours) sur les arbres ?

→ traitements dissociés : :Repertoire vs. :Fichier !!

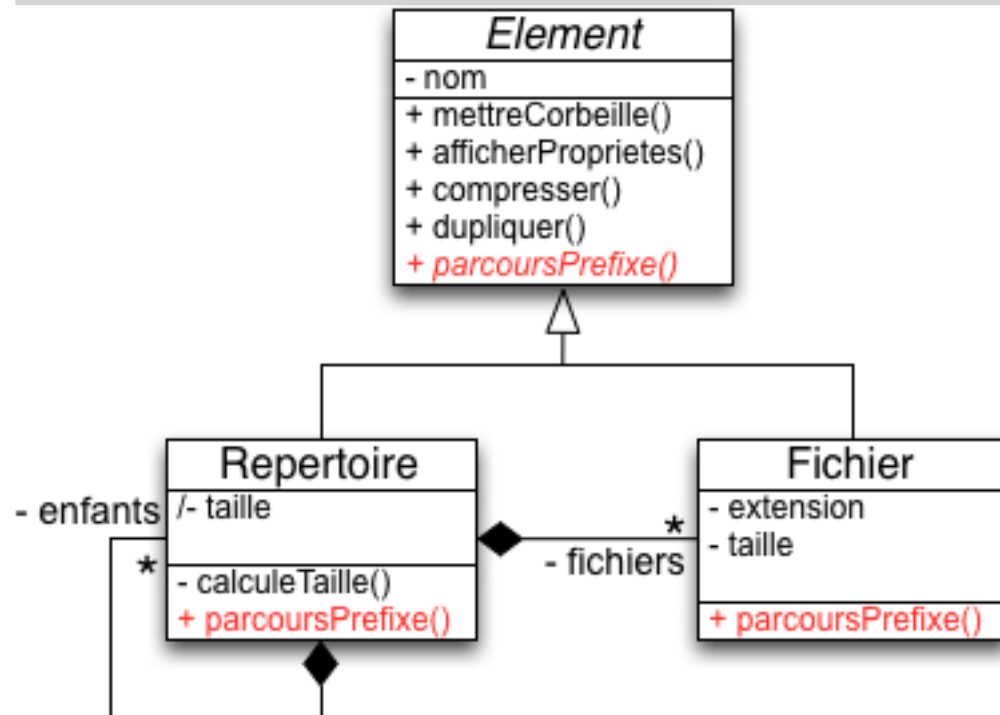


```
Void parcoursPrefixe(){
    traitement() ; //à remplacer (ex. affiche le nom)
    for(Repertoire r : enfants)
        r.parcoursPrefixe() ;
    for(Fichier f : fichiers)
        f.parcoursPrefixe();
}
```

```
Void parcoursPrefixe(){
    traitement() ; //à remplacer
    for(Arbre<T> e : enfants)
        e.parcoursPrefixe() ;
}
```

## A modéliser

*Améliorer la conception de façon à traiter tous les éléments d'un répertoire de manière uniforme.*



```

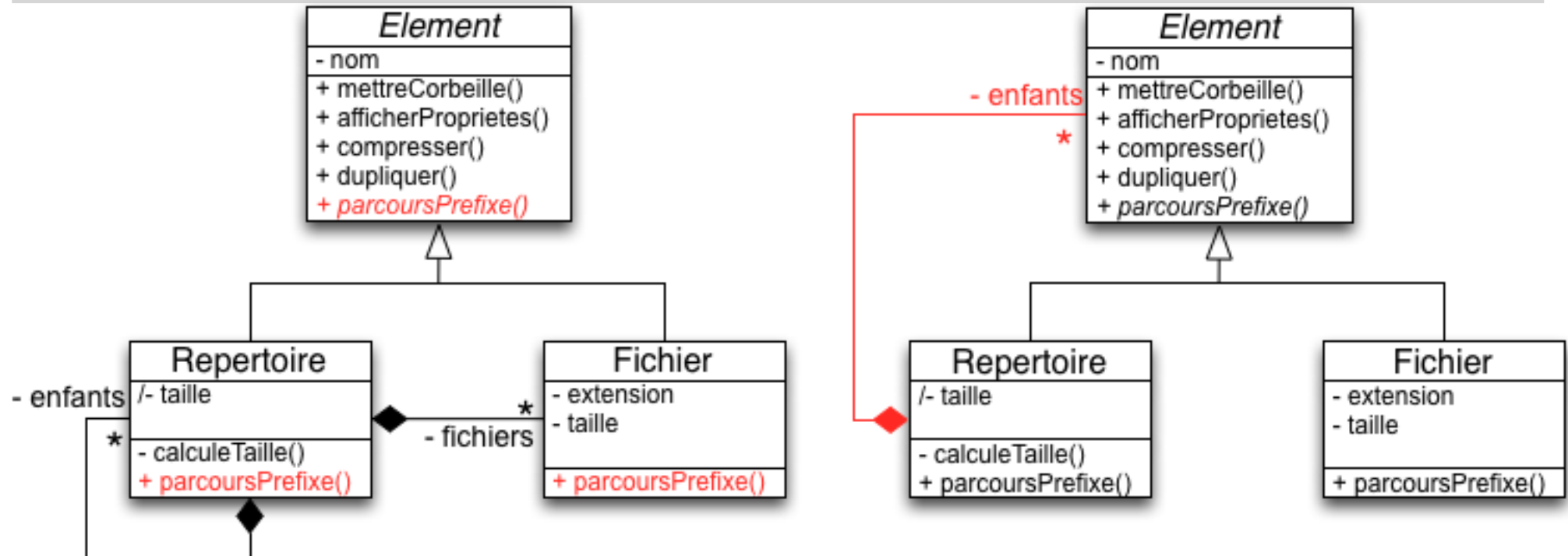
Void parcoursPrefixe(){
    traitement() ; //à remplacer (ex. affiche le nom)
    for(Repertoire r : enfants)
        r.parcoursPrefixe() ;
    for(Fichier f : fichiers)
        f.parcoursPrefixe();
}
  
```

```

Void parcoursPrefixe(){
    traitement() ; //à remplacer (ex. affiche le nom)
    for(Element e : enfants)
        e.parcoursPrefixe() ;
}
  
```

# A modéliser

*Améliorer la conception de façon à traiter tous les éléments d'un répertoire de manière uniforme.*



```

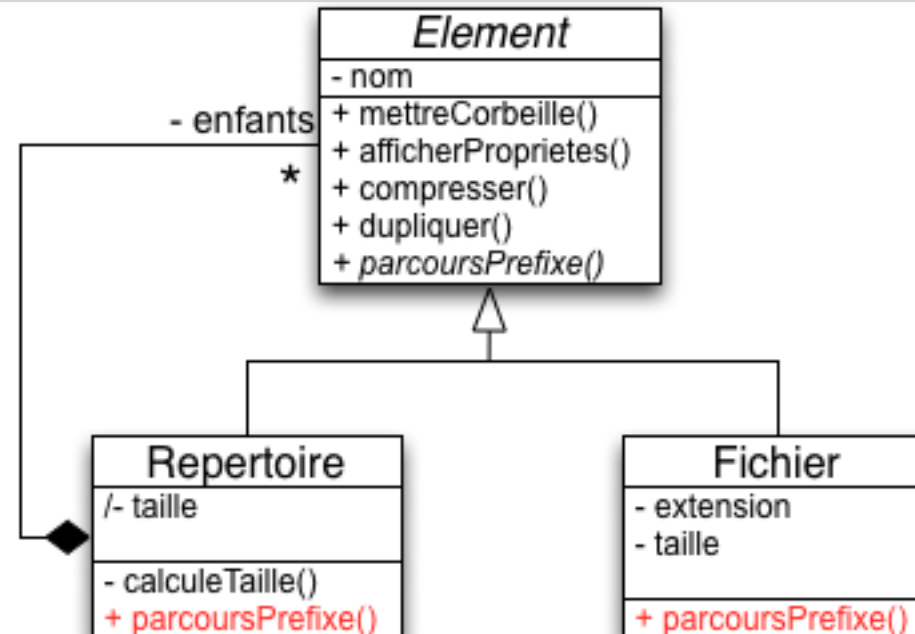
Void parcoursPrefixe(){
    traitement() ; //à remplacer (ex. affiche le nom)
    for(Repertoire r : enfants)
        r.parcoursPrefixe() ;
    for(Fichier f : fichiers)
        f.parcoursPrefixe();
}
  
```

```

Void parcoursPrefixe(){
    traitement() ; //à remplacer (ex. affiche le nom)
    for(Element e : enfants)
        e.parcoursPrefixe() ;
}
  
```

## A modéliser

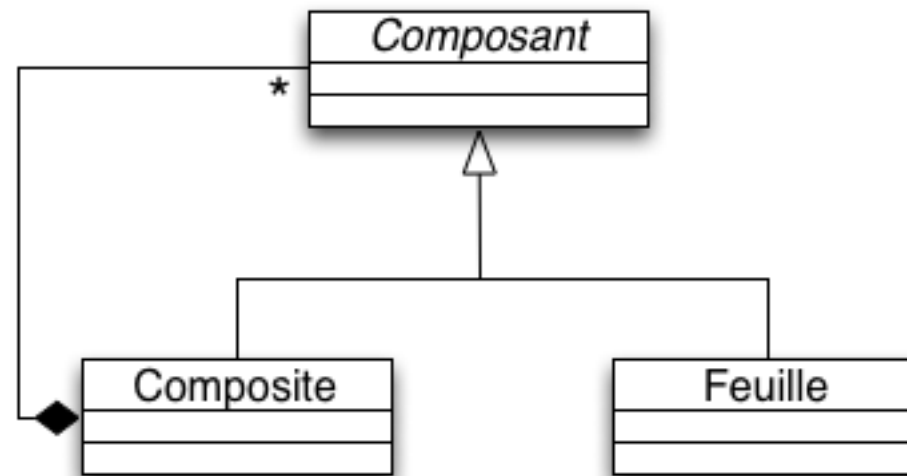
*Améliorer la conception de façon à traiter tous les éléments d'un répertoire de manière uniforme.*



```
//classe Repertoire
Void parcoursPrefixe(){
    traitement() ; //à remplacer (ex. affiche le nom)
    for(Element e : enfants)
        e.parcoursPrefixe();
}
```

```
//classe Fichier
Void parcoursPrefixe(){
    traitement() ; //à remplacer (ex. affiche le nom)
}
```

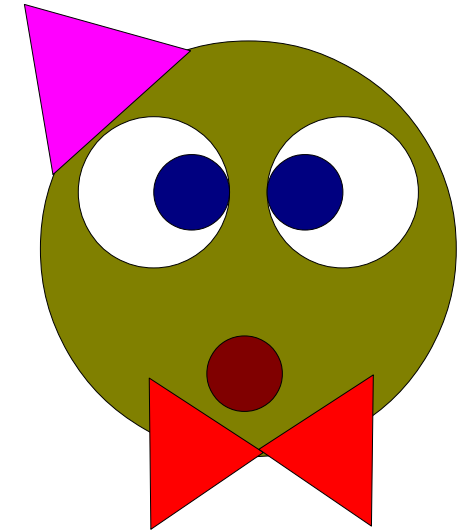
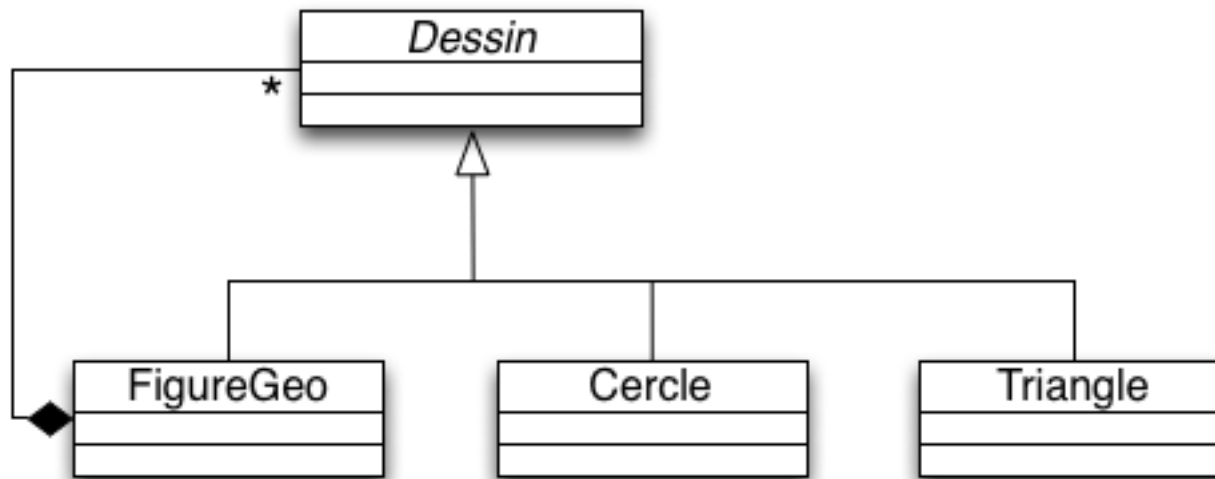
## Pattern « composite »



- Organiser des objets en **structure arborescente** (hiérarchie)
- Permettre la manipulation des feuilles et des nœuds de manière indifférenciée.



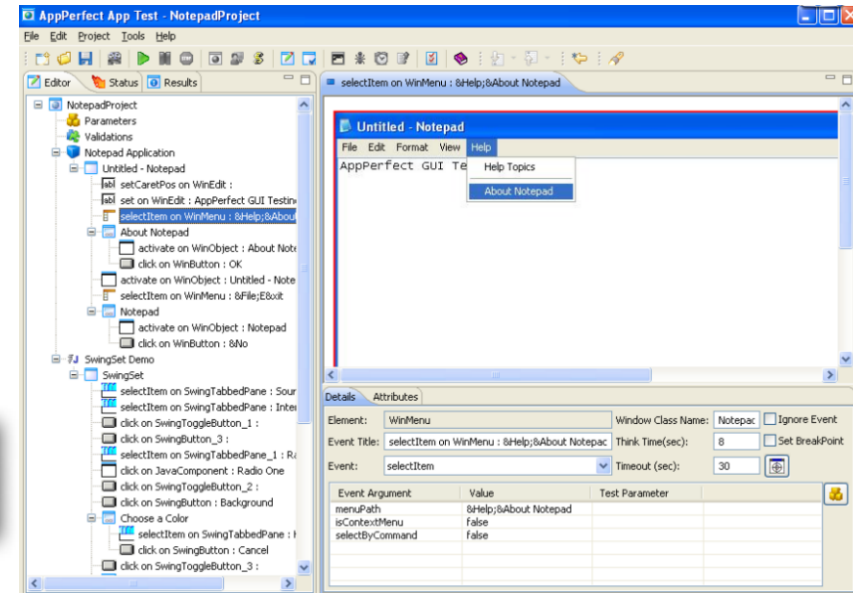
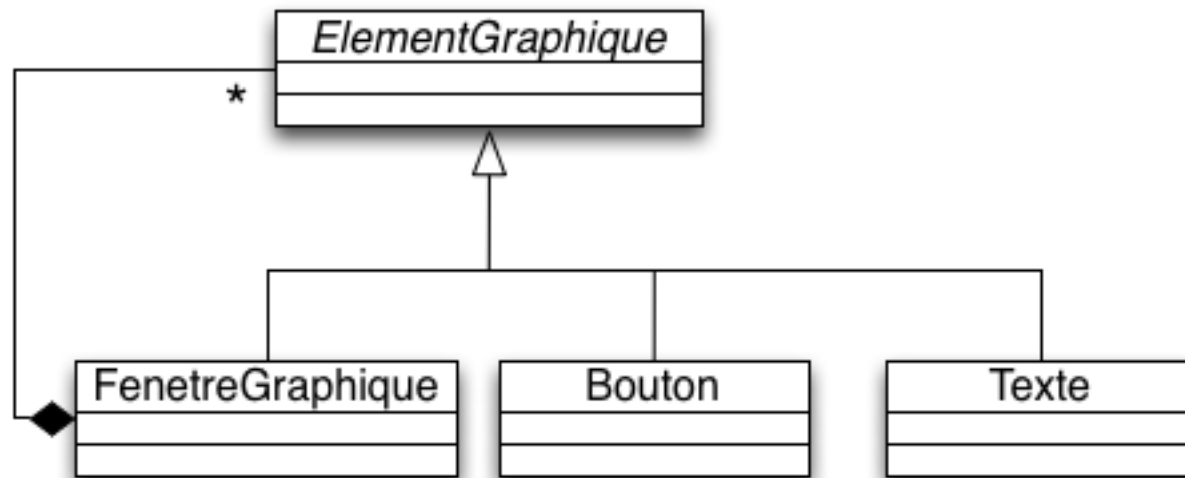
## Pattern « composite » : exemple



A modéliser

*Une figure géométrique est formée de sous-figures à base de cercles et de triangles.*

# Pattern « composite » : exemple

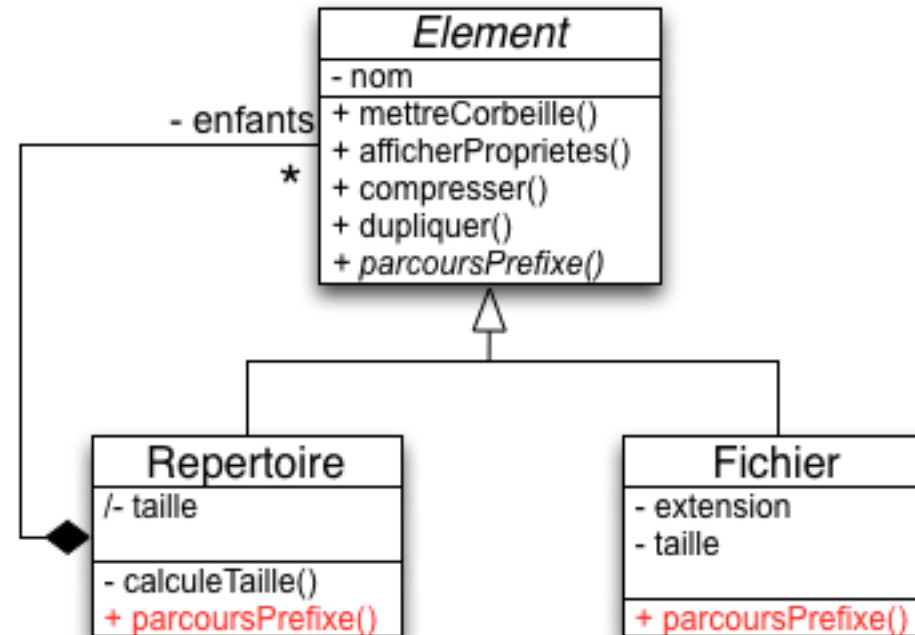


A modéliser

*Une fenêtre graphique est composée de fenêtres graphiques, d'images, de boutons, de textes, etc.*

# Modéliser une opération avec Un diagramme de collaboration

A modéliser

*Le fonctionnement de l'opération : parcoursPrefixe()*

//classe Repertoire

Void parcoursPrefixe(){

traitement() ; //à remplacer (ex. affiche le nom)

 for(**Element** e : enfants)
 e.parcoursPrefixe() ;

}

//classe Fichier

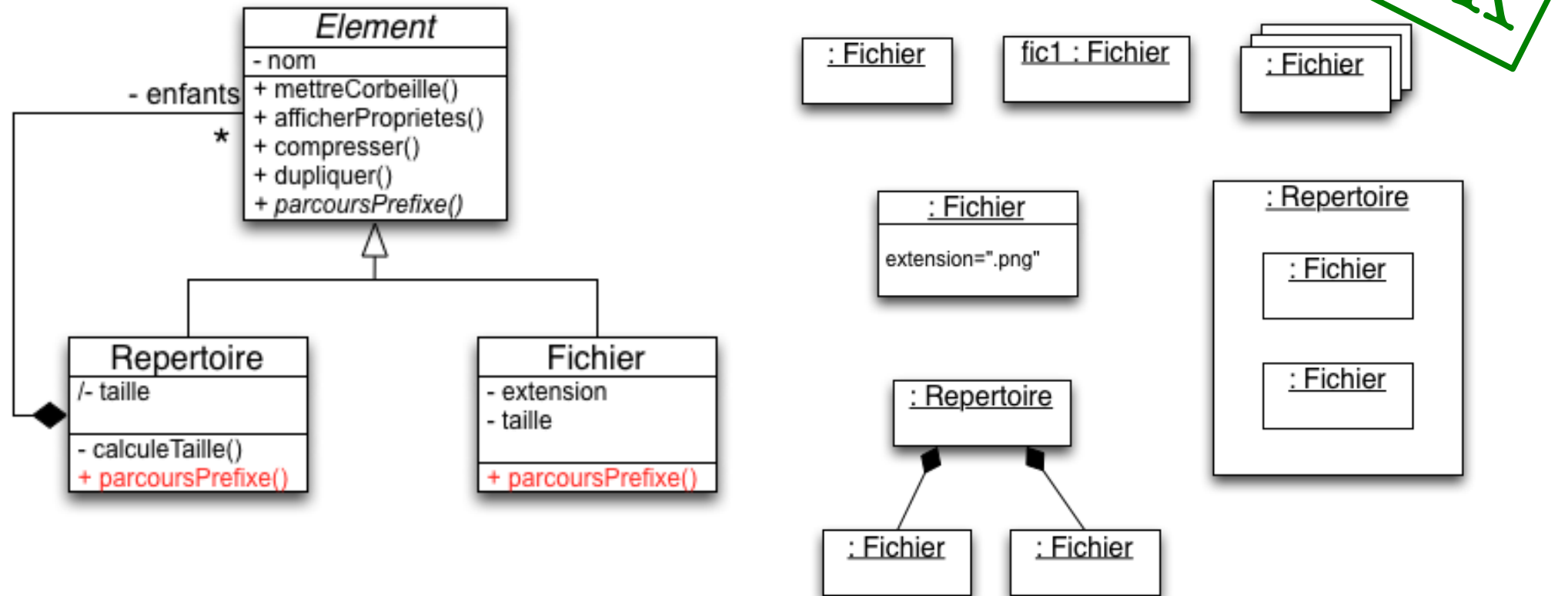
Void parcoursPrefixe(){

traitement() ; //à remplacer (ex. affiche le nom)

}

A modéliser

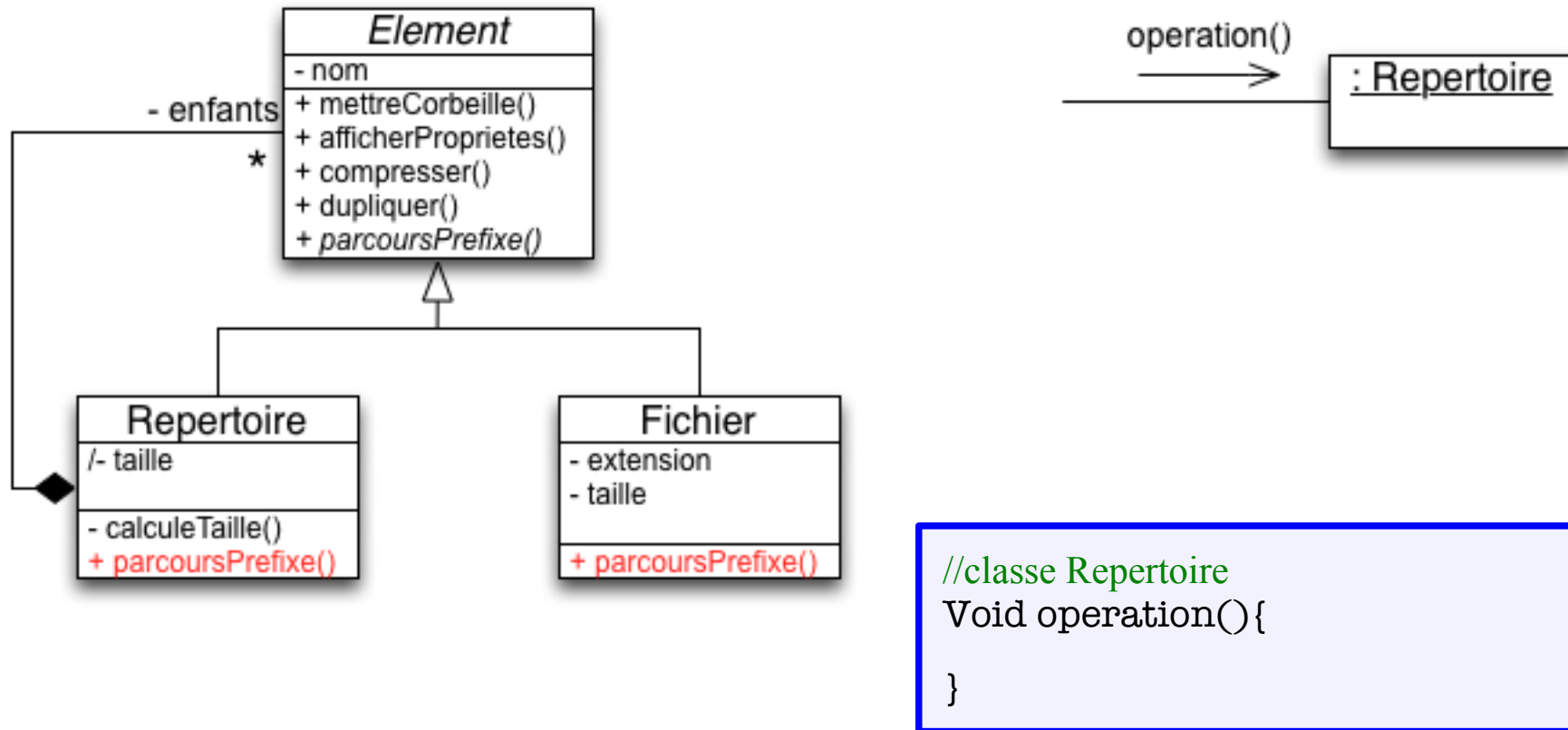
*Le fonctionnement de l'opération : `parcoursPrefixe()`*



Un diagramme de collaboration décrit la **dynamique** des **interactions** entre **objets** (instances des classes).

A modéliser

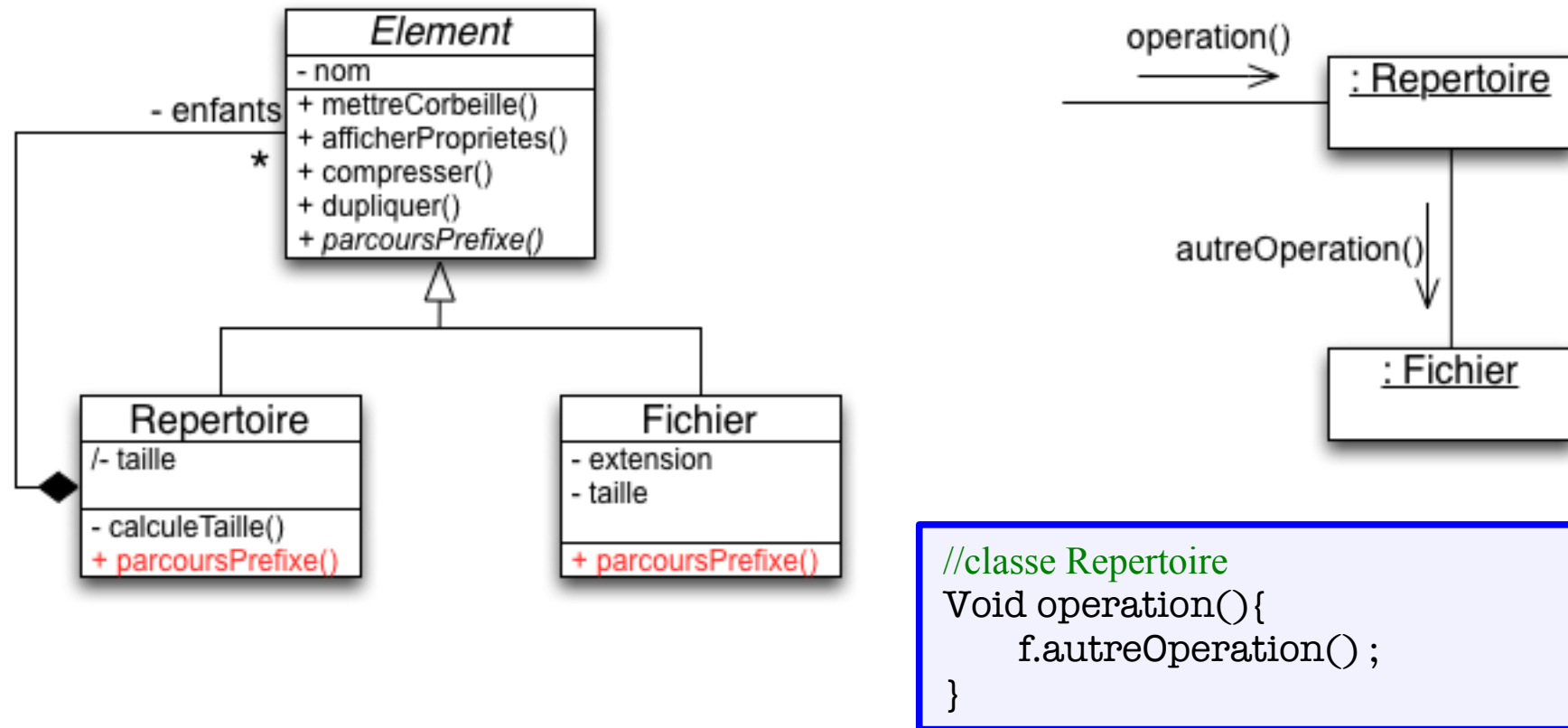
*Le fonctionnement de l'opération : `parcoursPrefixe()`*



Interaction = appel d'une opération sur un objet.  
Ici *operation()* est une opération de la classe : `Repertoire`

A modéliser

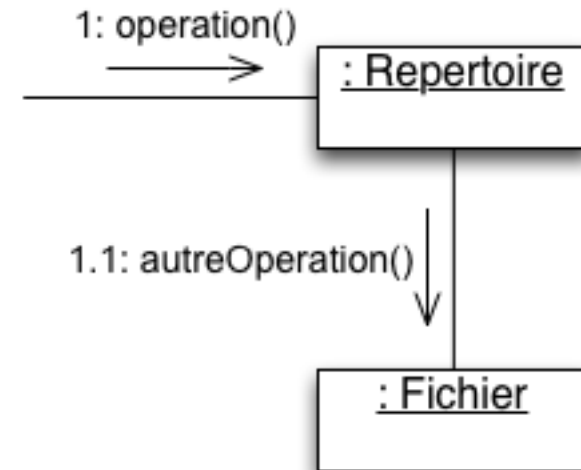
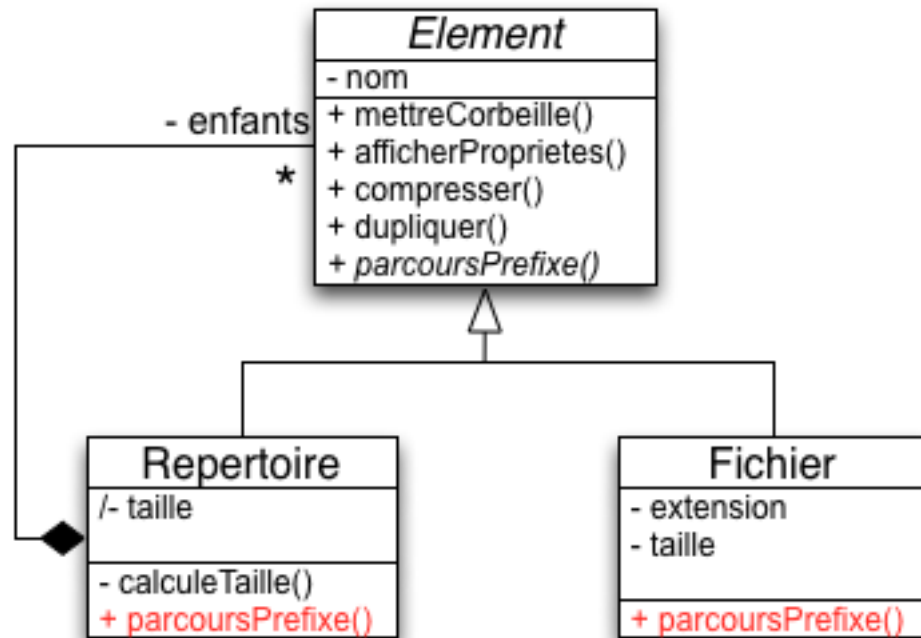
*Le fonctionnement de l'opération : `parcoursPrefixe()`*



Appels emboîtés

Ici *autreOperation()* est une opération de la classe : Fichier

A modéliser

*Le fonctionnement de l'opération : parcoursPrefixe()*

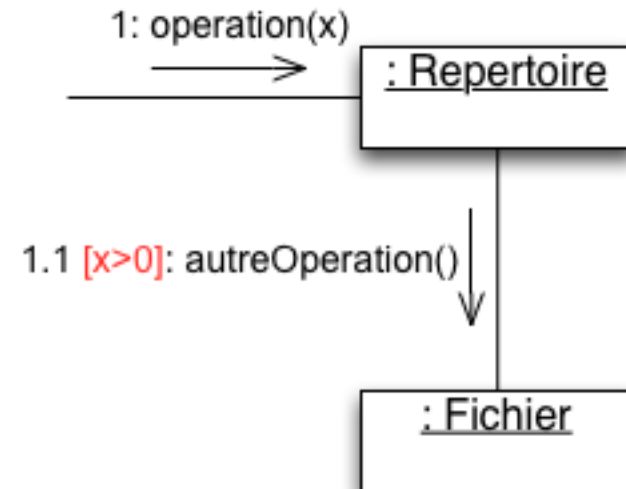
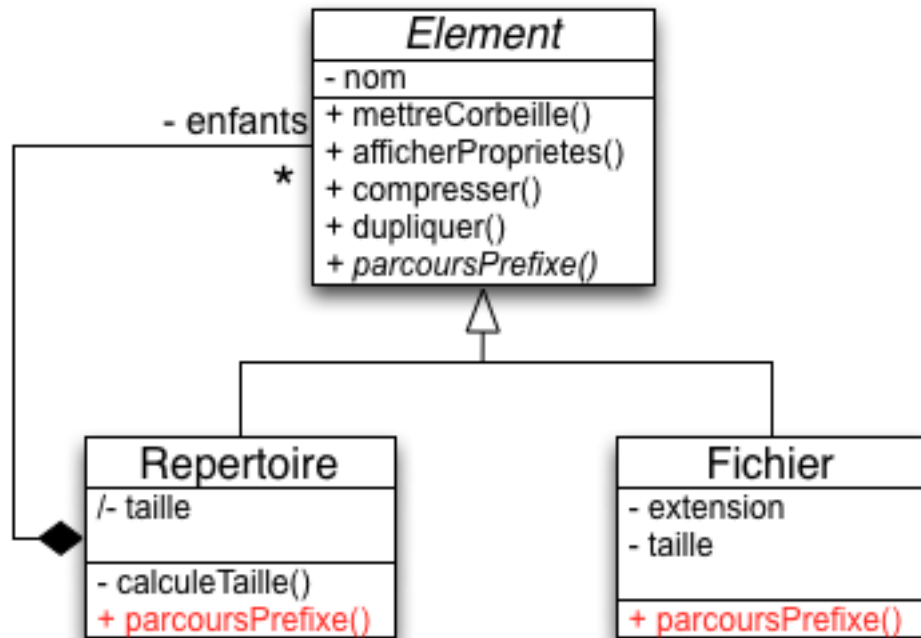
```

//classe Repertoire
Void operation(){
    f.autreOperation();
}
  
```

Appels emboîtés → **numéro de séquence**Ici *autreOperation()* est une opération de la classe : Fichier



A modéliser

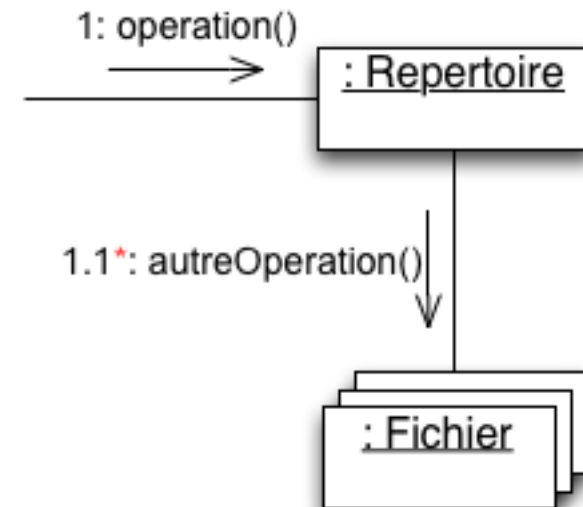
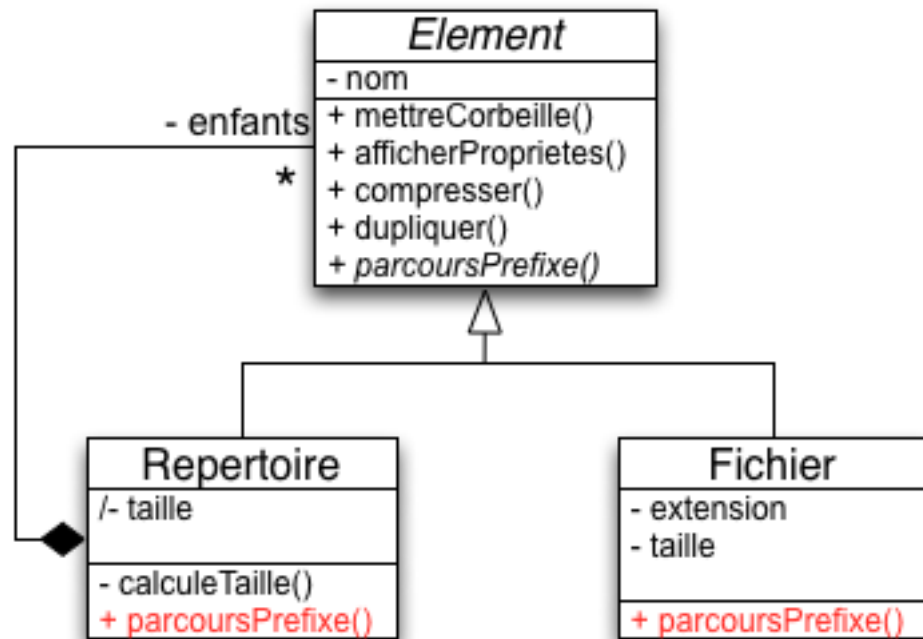
*Le fonctionnement de l'opération : parcoursPrefixe()*

```

Void operation(int x){
    if(x>0)
        f.autreOperation();
}
  
```

Appels conditionnés

A modéliser

*Le fonctionnement de l'opération : parcoursPrefixe()*

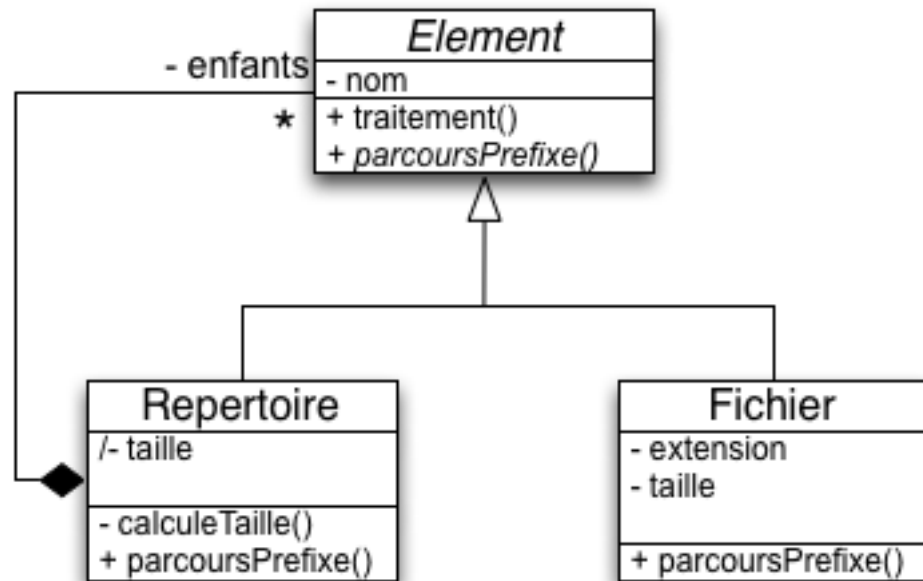
```

Void operation(){
    for(Element e : enfants)
        e.autreOperation();
    }
  
```

Appels multiples vers une collection d'objets

A modéliser

*Le fonctionnement de l'opération : `parcoursPrefixe()`*



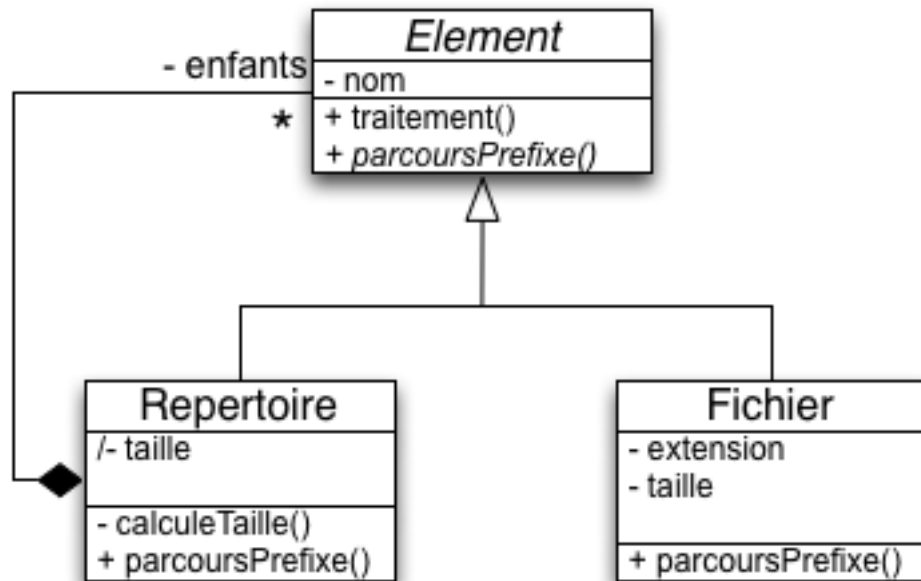
```

Void parcoursPrefixe() { //classe Repertoire
    traitement();
    for(Element e : enfants)
        e.parcoursPrefixe();
}
  
```

```

Void parcoursPrefixe() { //classe Fichier
    traitement();
}
  
```

A modéliser

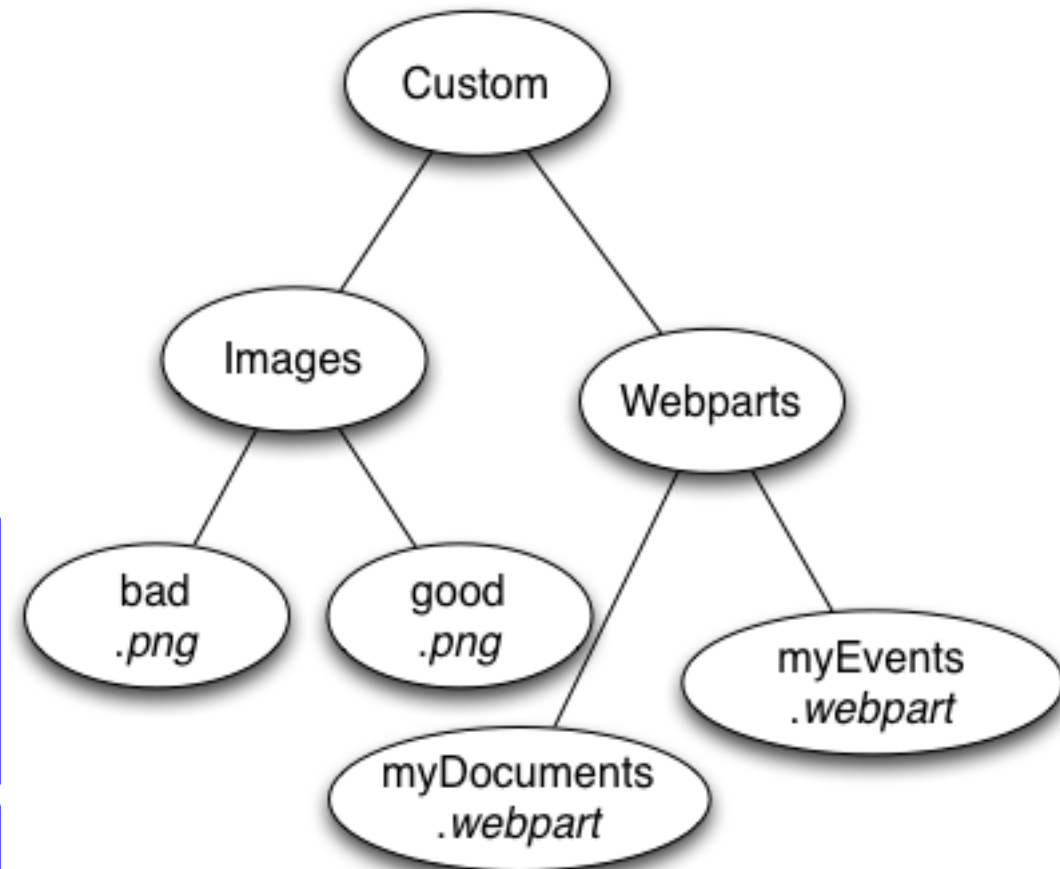
*Le fonctionnement de l'opération : parcoursPrefixe()*

```

Void parcoursPrefixe() { //classe Repertoire
    traitement();
    for(Element e : enfants)
        e.parcoursPrefixe();
}
  
```

```

Void parcoursPrefixe() { //classe Fichier
    traitement();
}
  
```



Questions ?