Structures de Données

TD1 – Introduction

Telecom Nancy

Olivier Festor - Mars 2022

Plan

Plan

Organisation

CM	3 ou 4x2h
TD/TP	2h de TD + 12h de TP = 32h de présence
Projet	$\approx$ 40h de travail personnel (sur volet Solveur)
Évaluation	1 écrit + 1 QCM + 1 TP noté

Les 4 objectifs principaux

Connaître et savoir **choisir**, **utiliser** et **évaluer** les structures de données usuelles.

Plan

Qu'est-ce que c'est ? À quoi ça sert ?

Qu'est-ce que c'est ? À quoi ça sert ?

Qu'est-ce que c'est ? À quoi ça sert ?

Une structure de données est une **organisation logique** des **données** qui vise à faciliter leur stockage et/ou leur traitement.

Qu'est-ce que c'est ? À quoi ça sert ?

Une structure de données est une **organisation logique** des **données** qui vise à faciliter leur stockage et/ou leur **traitement**.

Qu'est-ce que c'est ? À quoi ça sert ?

Une structure de données est une **organisation logique** des **données** qui vise à faciliter leur **stockage** et/ou leur traitement.

Qu'est-ce que c'est ? À quoi ça sert ?

Une structure de données est une **organisation logique** des **données** qui vise à faciliter leur **stockage** et/ou leur **traitement**.

Et en informatique ?

Quelles sont les structures de données usuelles ?

Et en informatique ?

Quelles sont les structures de données usuelles ?

Les structures linéaires les vecteurs, les listes, les piles, les files, les tables...

Les structures arborescentes les arbres : binaires, AVL, rouge/noir...

Les structures relationnelles les graphes : orientés, non orientés...

Et en informatique ?

Quelles sont les structures de données usuelles ?

Les structures linéaires les vecteurs, les listes, les piles, les files, les tables...

Les structures arborescentes les arbres : binaires, AVL, rouge/noir...

Les structures relationnelles les graphes : orientés, non orientés...

Ce sont celles que nous allons voir !

Et en informatique ?

C'est quoi optimiser le stockage ?

Et en informatique ?

C'est quoi optimiser le stockage ?

$\rightarrow$ réduire l'occupation en mémoire (RAM)

Et en informatique ?

C'est quoi optimiser le stockage ?

$\rightarrow$ réduire l'occupation en mémoire (RAM)

C'est quoi optimiser le traitement ?

Et en informatique ?

C'est quoi optimiser le stockage ?

$\rightarrow$ réduire l'occupation en mémoire (RAM)

C'est quoi optimiser le traitement ?

$\rightarrow$ réduire le temps d'accès aux données

Comment allons-nous procéder ?

Pour chaque structure de donnée :

1. Spécification algébrique (au début)
2. Implantation (en Java pour l'instant)
3. Algorithmes associés (recherche, ajout, suppression, . . .)
4. Analyse de complexité

Plan

Type abstrait

Un type est dit **abstrait** s'il ne tient pas compte de la **représentation** (organisation) de ses données.

Type abstrait

Un type est dit **abstrait** s'il ne tient pas compte de la **représentation** (organisation) de ses données.

Exemple (quotidien)

Une bibliothèque.

Type abstrait

Un type est dit **abstrait** s'il ne tient pas compte de la **représentation** (organisation) de ses données.

Exemple (quotidien)

Une bibliothèque.

Exemple (Java)

```
Collection<Livres> maBib; // Abstrait
PriorityQueue<Livres> maBibClassee; // Non abstrait
```

Type abstrait

Un type est dit **abstrait** s'il ne tient pas compte de la **représentation** (organisation) de ses données.

Exemple (quotidien)

Une bibliothèque.

Exemple (Java)

```
Collection<Livres> maBib; // Abstrait
PriorityQueue<Livres> maBibClassee; // Non abstrait
```

Un **type abstrait** est souvent associé à une **interface**.

Généricité

On parle de **type générique** lorsqu'il ne spécifie pas le **type des données** traitées.

Généricité

On parle de **type générique** lorsqu'il ne spécifie pas le **type des données** traitées.

Exemple (quotidien)

Une pile.

Généricité

On parle de **type générique** lorsqu'il ne spécifie pas le **type des données** traitées.

Exemple (quotidien)

Une pile.

Exemple (Java)

```
Collection<E> mesDonnees; // Générique
Collection<Musiques> mesMusiques; // Pas générique
```

Généricité

On parle de **type générique** lorsqu'il ne spécifie pas le **type des données** traitées.

Exemple (quotidien)

Une pile.

Exemple (Java)

```
Collection<E> mesDonnees; // Générique
Collection<Musiques> mesMusiques; // Pas générique
```

On obtient un **type effectif** lorqu'on **instancie** un type générique.

Spécification algébrique

On parle de **spécification algébrique** d'un type abstrait lorsqu'on le définit par :

- une **signature** (les opérations du type)
- des **axiomes** (les propriétés de ses opérations)

Spécification algébrique

On parle de **spécification algébrique** d'un type abstrait lorsqu'on le définit par :

- une **signature** (les opérations du type)
- des **axiomes** (les propriétés de ses opérations)

Attention !

Une spécification algébrique ne définit pas le type des données !

Présentation

Type (myst)R

Opérations :

$$
\begin{array}{rll}
m : & T \times T & \rightarrow R \\
p : & R \times T \times S & \rightarrow R \\
o : & R \times T & \rightarrow S \\
q : & R & \rightarrow T \\
v : & R & \rightarrow T
\end{array}
$$

Présentation

Type Vecteur

Opérations :

$$\begin{aligned}
\textit{nouveau} &: \quad \textit{Entier} \times \textit{Entier} && \rightarrow \textit{Vecteur}\\
\textit{changer-ieme} &: \quad \textit{Vecteur} \times \textit{Entier} \times \textit{Element} && \rightarrow \textit{Vecteur}\\
\textit{ieme} &: \quad \textit{Vecteur} \times \textit{Entier} && \rightarrow \textit{Element}\\
\textit{indice.sup} &: \quad \textit{Vecteur} && \rightarrow \textit{Entier}\\
\textit{indice.inf} &: \quad \textit{Vecteur} && \rightarrow \textit{Entier}
\end{aligned}$$

Présentation

Type Vecteur

Opérations :

$$\begin{aligned}
\textit{nouveau} &: \quad \textit{Entier} \times \textit{Entier} && \rightarrow \textit{Vecteur} \\
\textit{changer-ieme} &: \quad \textit{Vecteur} \times \textit{Entier} \times \textit{Element} && \rightarrow \textit{Vecteur} \\
\textit{ieme} &: \quad \textit{Vecteur} \times \textit{Entier} && \rightarrow \textit{Element} \\
\textit{indice.sup} &: \quad \textit{Vecteur} && \rightarrow \textit{Entier} \\
\textit{indice.inf} &: \quad \textit{Vecteur} && \rightarrow \textit{Entier}
\end{aligned}$$

Profil d'une opération :

$$\text{nom : type arg } 1 \times \ldots \times \text{ type arg n} \rightarrow \text{type retour}$$

Présentation

Type Vecteur

Opérations :

$$\begin{array}{rlll} \textit{nouveau} : & \textit{Entier} \times \textit{Entier} & \to & \textit{Vecteur} \\ \textit{changer-ieme} : & \textit{Vecteur} \times \textit{Entier} \times \textit{Element} & \to & \textit{Vecteur} \\ \textit{ieme} : & \textit{Vecteur} \times \textit{Entier} & \to & \textit{Element} \\ \textit{indice.sup} : & \textit{Vecteur} & \to & \textit{Entier} \\ \textit{indice.inf} : & \textit{Vecteur} & \to & \textit{Entier} \end{array}$$

Profil d'une opération :

nom : type arg 1 $\times \ldots \times$ type arg n $\to$ type retour

Présentation

Type Vecteur

Opérations :

$$
\begin{aligned}
\textit{nouveau} &: \quad \textit{Entier} \times \textit{Entier} && \rightarrow \textit{Vecteur} \\
\textit{changer-ieme} &: \quad \textit{Vecteur} \times \textit{Entier} \times \textit{Element} && \rightarrow \textit{Vecteur} \\
\textit{ieme} &: \quad \textit{Vecteur} \times \textit{Entier} && \rightarrow \textit{Element} \\
\textit{indice.sup} &: \quad \textit{Vecteur} && \rightarrow \textit{Entier} \\
\textit{indice.inf} &: \quad \textit{Vecteur} && \rightarrow \textit{Entier}
\end{aligned}
$$

Profil d'une opération :

$$\text{nom} : \text{type arg 1} \times \ldots \times \text{type arg n} \rightarrow \text{type retour}$$

Présentation

Type Vecteur

Opérations :

$$\begin{array}{rll} \textit{nouveau}: & \textit{Entier} \times \textit{Entier} & \rightarrow \textit{Vecteur} \\ \textit{changer-ieme}: & \textit{Vecteur} \times \textit{Entier} \times \textit{Element} & \rightarrow \textit{Vecteur} \\ \textit{ieme}: & \textit{Vecteur} \times \textit{Entier} & \rightarrow \textit{Element} \\ \textit{indice.sup}: & \textit{Vecteur} & \rightarrow \textit{Entier} \\ \textit{indice.inf}: & \textit{Vecteur} & \rightarrow \textit{Entier} \end{array}$$

Profil d'une opération :

nom : type arg 1 $\times$ ... $\times$ type arg n $\rightarrow$ type retour

Présentation

Type Vecteur

Opérations :

$$
\begin{aligned}
\textit{nouveau} &: \textit{Entier} \times \textit{Entier} && \rightarrow \textit{Vecteur} \\
\textit{changer-ieme} &: \textit{Vecteur} \times \textit{Entier} \times \textit{Element} && \rightarrow \textit{Vecteur} \\
\textit{ieme} &: \textit{Vecteur} \times \textit{Entier} && \rightarrow \textit{Element} \\
\textit{indice.sup} &: \textit{Vecteur} && \rightarrow \textit{Entier} \\
\textit{indice.inf} &: \textit{Vecteur} && \rightarrow \textit{Entier}
\end{aligned}
$$

Profil d'une opération :

$$\text{nom : type arg } 1 \times \ldots \times \text{ type arg n} \rightarrow \text{type retour}$$

En programmation, on parle de profil de fonction.

Classification des opérations

Type Vecteur

Opérations :

$$
\begin{array}{rlr}
\textit{nouveau}: & \textit{Entier} \times \textit{Entier} & \rightarrow \textit{Vecteur} \\
\textit{changer-ieme}: & \textit{Vecteur} \times \textit{Entier} \times \textit{Element} & \rightarrow \textit{Vecteur} \\
\textit{ieme}: & \textit{Vecteur} \times \textit{Entier} & \rightarrow \textit{Element} \\
\textit{indice.sup}: & \textit{Vecteur} & \rightarrow \textit{Entier} \\
\textit{indice.inf}: & \textit{Vecteur} & \rightarrow \textit{Entier}
\end{array}
$$

On peut classer les opérations en deux sous-ensembles :

Classification des opérations

Type Vecteur

Opérations :

$$\begin{array}{rll} \textit{nouveau}: & \textit{Entier} \times \textit{Entier} & \rightarrow \textit{Vecteur} \\ \textit{changer-ieme}: & \textit{Vecteur} \times \textit{Entier} \times \textit{Element} & \rightarrow \textit{Vecteur} \\ \textit{ieme}: & \textit{Vecteur} \times \textit{Entier} & \rightarrow \textit{Element} \\ \textit{indice.sup}: & \textit{Vecteur} & \rightarrow \textit{Entier} \\ \textit{indice.inf}: & \textit{Vecteur} & \rightarrow \textit{Entier} \end{array}$$

On peut classer les opérations en deux sous-ensembles :

- Les **opérations internes**

Classification des opérations

Type Vecteur

Opérations :

$$
\begin{array}{rll}
\textit{nouveau}: & \textit{Entier} \times \textit{Entier} & \rightarrow \textit{Vecteur} \\
\textit{changer-ieme}: & \textit{Vecteur} \times \textit{Entier} \times \textit{Element} & \rightarrow \textit{Vecteur} \\
\textit{ieme}: & \textit{Vecteur} \times \textit{Entier} & \rightarrow \textit{Element} \\
\textit{indice.sup}: & \textit{Vecteur} & \rightarrow \textit{Entier} \\
\textit{indice.inf}: & \textit{Vecteur} & \rightarrow \textit{Entier}
\end{array}
$$

On peut classer les opérations en deux sous-ensembles :

- Les **opérations internes**
- Les opérations d'observation (**observateurs**)

Classification des opérations

Type Vecteur

Opérations :

$$
\begin{array}{rlr}
\textit{nouveau} : & \textit{Entier} \times \textit{Entier} & \rightarrow \textit{Vecteur} \\
\textit{changer-ieme} : & \textit{Vecteur} \times \textit{Entier} \times \textit{Element} & \rightarrow \textit{Vecteur} \\
\textit{ieme} : & \textit{Vecteur} \times \textit{Entier} & \rightarrow \textit{Element} \\
\textit{indice.sup} : & \textit{Vecteur} & \rightarrow \textit{Entier} \\
\textit{indice.inf} : & \textit{Vecteur} & \rightarrow \textit{Entier}
\end{array}
$$

On peut classer les opérations en deux sous-ensembles :

- Les **opérations internes**
- Les opérations d'observation (**observateurs**)

Remarque

Seul le **constructeur** n'a pas de paramètre du type défini.

Généralisation

Type (Myst)R

Opérations :

$$\begin{array}{lll} m: & T \times T & \rightarrow R \\ p: & R \times T \times S & \rightarrow R \\ o: & R \times T & \rightarrow S \\ q: & R & \rightarrow T \\ v: & R & \rightarrow T \end{array}$$

Généralisation

Type (Myst)R

Opérations :

$$\begin{array}{rll}
m: & T \times T & \rightarrow R \\
p: & R \times T \times S & \rightarrow R \\
o: & R \times T & \rightarrow S \\
q: & R & \rightarrow T \\
v: & R & \rightarrow T
\end{array}$$

Type Vecteur

Opérations :

$$\begin{array}{rll}
\mathit{nouveau}: & \mathit{Entier} \times \mathit{Entier} & \rightarrow \mathit{Vecteur} \\
\mathit{changer\text{-}ieme}: & \mathit{Vecteur} \times \mathit{Entier} \times \mathit{Element} & \rightarrow \mathit{Vecteur} \\
\mathit{ieme}: & \mathit{Vecteur} \times \mathit{Entier} & \rightarrow \mathit{Element} \\
\mathit{indice.sup}: & \mathit{Vecteur} & \rightarrow \mathit{Entier} \\
\mathit{indice.inf}: & \mathit{Vecteur} & \rightarrow \mathit{Entier}
\end{array}$$

Généralisation

Type (Myst)R

Opérations :

$$\begin{array}{rll} m: & T \times T & \to R \\ p: & R \times T \times S & \to R \\ o: & R \times T & \to S \\ q: & R & \to T \\ v: & R & \to T \end{array}$$

- Les types Vecteur et R ont la **même signature**

Généralisation

Type (Myst)R

Opérations :

$$
\begin{array}{rll}
m: & T \times T & \rightarrow R \\
p: & R \times T \times S & \rightarrow R \\
o: & R \times T & \rightarrow S \\
q: & R & \rightarrow T \\
v: & R & \rightarrow T
\end{array}
$$

- Les types Vecteur et R ont la **même signature**
- La **compréhension** du type Vecteur est basée sur l'intuition du lecteur $\rightarrow$ c'est la **sémantique** ($=$ sens) du type

Généralisation

Type (Myst)R

Opérations :

$$
\begin{array}{rll}
m: & T \times T & \rightarrow R \\
p: & R \times T \times S & \rightarrow R \\
o: & R \times T & \rightarrow S \\
q: & R & \rightarrow T \\
v: & R & \rightarrow T
\end{array}
$$

- Les types Vecteur et R ont la **même signature**
- La **compréhension** du type Vecteur est basée sur l'intuition du lecteur → c'est la **sémantique** (= sens) du type

Comment définir la sémantique d'un type ?

Présentation

Les **axiomes** :

- permettent de donner la **sémantique** des opérations
- correspondent aux **propriétés** des opérations

Type Vecteur

Axiomes :

$$\ldots$$
$$\mathit{ieme}(\mathit{changer\text{-}ieme}(v, i, e), i) = e$$
$$\ldots$$

où v est de type Vecteur, i de type Entier et e de type Element.

Exemple

Type Vecteur

Axiomes :

$$
\begin{aligned}
ieme(nouveau(i,j),k) &= 0 \\
ieme(changer\text{-}ieme(v,i,e),j) &= \begin{cases} e & \text{si } i = j \\ ieme(v,j) & \text{si } i \neq j \end{cases} \\
indice.inf(nouveau(i,j)) &= i \\
indice.inf(changer\text{-}ieme(v,i,e)) &= indice.inf(v) \\
indice.sup(nouveau(i,j)) &= j \\
indice.sup(changer\text{-}ieme(v,i,e)) &= indice.sup(v)
\end{aligned}
$$

avec v de type Vecteur, i, j, k de type Entier et e de type Element.

Exemple

Type Vecteur

Axiomes :

$$
\begin{array}{rcl}
ieme(nouveau(i,j),k) & = & 0 \\
ieme(changer\text{-}ieme(v,i,e),j) & = & \begin{cases} e & \text{si } i = j \\ ieme(v,j) & \text{si } i \neq j \end{cases} \\
indice.inf(nouveau(i,j)) & = & i \\
indice.inf(changer\text{-}ieme(v,i,e)) & = & indice.inf(v) \\
indice.sup(nouveau(i,j)) & = & j \\
indice.sup(changer\text{-}ieme(v,i,e)) & = & indice.sup(v)
\end{array}
$$

avec v de type Vecteur, i, j, k de type Entier et e de type Element.

Énoncer ainsi les axiomes n'est pas suffisant.

Les préconditions d'axiomes

Il est nécessaire d'ajouter des **préconditions** sur les variables.

Type Vecteur

Préconditions :

$$\begin{array}{rll} nouveau(i,j) & defini\ ssi & 0 \leq i \leq j \\ changer\text{-}ieme(v,i,e) & defini\ ssi & indice.inf(v) \leq i \leq indice.sup(v) \\ ieme(v,i) & defini\ ssi & indice.inf(v) \leq i \leq indice.sup(v) \end{array}$$

Cohérence et complétude

Deux questions « existentielles » :

- Y'a-t-il des axiomes **contradictoires** ?

Cohérence et complétude

Deux questions « existentielles » :

- Y'a-t-il des axiomes **contradictoires** ? $\rightarrow$ **cohérence**

Cohérence et complétude

Deux questions « existentielles » :

- Y'a-t-il des axiomes **contradictoires** ? $\rightarrow$ **cohérence**
- Y'a-t-il **suffisamment** d'axiomes ?

Cohérence et complétude

Deux questions « existentielles » :

- Y'a-t-il des axiomes **contradictoires** ? → **cohérence**
- Y'a-t-il **suffisamment** d'axiomes ? → **complétude**

Cohérence et complétude

Deux questions « existentielles » :

- Y'a-t-il des axiomes **contradictoires** ? $\rightarrow$ **cohérence**
- Y'a-t-il **suffisamment** d'axiomes ? $\rightarrow$ **complétude**

Les axiomes DOIVENT être cohérents ET complets.

S'assurer de la cohérence et de la complétude

Il faut écrire les axiomes qui définissent le résultat de la composition de tous les observateurs avec toutes les opérations internes.

Exemple

Observateurs	Opérations internes
$ieme : V \times I \to E$	$nouveau : I \times I \to V$
$indice.inf : V \to E$	$changer\text{-}ieme : V \times I \times E \to V$
$indice.sup : V \to E$	

Axiomes :

$$ieme(nouveau(i,j),k) = 0$$
$$ieme(changer\text{-}ieme(v,i,e),j) = e \text{ si } i = j, ieme(v,j) \text{ si } i \neq j$$
$$indice.inf(nouveau(i,j)) = i$$
$$indice.inf(changer\text{-}ieme(v,i,e)) = indice.inf(v)$$
$$indice.sup(nouveau(i,j)) = j$$
$$indice.sup(changer\text{-}ieme(v,i,e)) = indice.sup(v)$$

S'assurer de la cohérence et de la complétude

Il faut écrire les axiomes qui définissent le résultat de la composition de tous les observateurs avec toutes les opérations internes.

Exemple

Observateurs	Opérations internes
$ieme : V \times I \rightarrow E$	$nouveau : I \times I \rightarrow V$
$indice.inf : V \rightarrow E$	$changer\text{-}ieme : V \times I \times E \rightarrow V$
$indice.sup : V \rightarrow E$	

Axiomes :

$ieme(nouveau(i, j), k) = 0$
$ieme(changer\text{-}ieme(v, i, e), j) = e$ si $i = j$, $ieme(v, j)$ si $i \neq j$
$indice.inf(nouveau(i, j)) = i$
$indice.inf(changer\text{-}ieme(v, i, e)) = indice.inf(v)$
$indice.sup(nouveau(i, j)) = j$
$indice.sup(changer\text{-}ieme(v, i, e)) = indice.sup(v)$

S'assurer de la cohérence et de la complétude

Il faut écrire les axiomes qui définissent le résultat de la composition de tous les observateurs avec toutes les opérations internes.

Exemple

Observateurs	Opérations internes
$ieme : V \times I \to E$	$nouveau : I \times I \to V$
$indice.inf : V \to E$	$changer\text{-}ieme : V \times I \times E \to V$
$indice.sup : V \to E$	

Axiomes :

$ieme(nouveau(i,j),k) = 0$

$ieme(changer\text{-}ieme(v,i,e),j) = e \text{ si } i = j, ieme(v,j) \text{ si } i \neq j$

$indice.inf(nouveau(i,j)) = i$

$indice.inf(changer\text{-}ieme(v,i,e)) = indice.inf(v)$

$indice.sup(nouveau(i,j)) = j$

$indice.sup(changer\text{-}ieme(v,i,e)) = indice.sup(v)$

S'assurer de la cohérence et de la complétude

Il faut écrire les axiomes qui définissent le résultat de la composition de tous les observateurs avec toutes les opérations internes.

Exemple

Observateurs	Opérations internes
$ieme : V \times I \to E$	$nouveau : I \times I \to V$
$indice.inf : V \to E$	$changer\text{-}ieme : V \times I \times E \to V$
$indice.sup : V \to E$	

Axiomes :

$ieme(nouveau(i,j),k) = 0$

$ieme(changer\text{-}ieme(v,i,e),j) = e$ si $i = j$, $ieme(v,j)$ si $i \neq j$

$indice.inf(nouveau(i,j)) = i$

$indice.inf(changer\text{-}ieme(v,i,e)) = indice.inf(v)$

$indice.sup(nouveau(i,j)) = j$

$indice.sup(changer\text{-}ieme(v,i,e)) = indice.sup(v)$

S'assurer de la cohérence et de la complétude

Il faut écrire les axiomes qui définissent le résultat de la composition de tous les observateurs avec toutes les opérations internes.

Exemple

Observateurs	Opérations internes
$ieme : V \times I \to E$	$nouveau : I \times I \to V$
$indice.inf : V \to E$	$changer\text{-}ieme : V \times I \times E \to V$
$indice.sup : V \to E$	

Axiomes :

$ieme(nouveau(i,j),k) = 0$

$ieme(changer\text{-}ieme(v,i,e),j) = e$ si $i = j$, $ieme(v,j)$ si $i \neq j$

$indice.inf(nouveau(i,j)) = i$

$indice.inf(changer\text{-}ieme(v,i,e)) = indice.inf(v)$

$indice.sup(nouveau(i,j)) = j$

$indice.sup(changer\text{-}ieme(v,i,e)) = indice.sup(v)$

S'assurer de la cohérence et de la complétude

Il faut écrire les axiomes qui définissent le résultat de la composition de tous les observateurs avec toutes les opérations internes.

Exemple

Observateurs	Opérations internes
$ieme : V \times I \to E$	$nouveau : I \times I \to V$
$indice.inf : V \to E$	$changer\text{-}ieme : V \times I \times E \to V$
$indice.sup : V \to E$	

Axiomes :

$ieme(nouveau(i,j),k) = 0$

$ieme(changer\text{-}ieme(v,i,e),j) = e$ si $i = j$, $ieme(v,j)$ si $i \neq j$

$indice.inf(nouveau(i,j)) = i$

$indice.inf(changer\text{-}ieme(v,i,e)) = indice.inf(v)$

$indice.sup(nouveau(i,j)) = j$

$indice.sup(changer\text{-}ieme(v,i,e)) = indice.sup(v)$

S'assurer de la cohérence et de la complétude

Il faut écrire les axiomes qui définissent le résultat de la composition de tous les observateurs avec toutes les opérations internes.

Exemple

Observateurs	Opérations internes
$ieme : V \times I \to E$	$nouveau : I \times I \to V$
$indice.inf : V \to E$	$changer\text{-}ieme : V \times I \times E \to V$
$indice.sup : V \to E$	

Axiomes :

$ieme(nouveau(i,j),k) = 0$

$ieme(changer\text{-}ieme(v,i,e),j) = e$ si $i = j$, $ieme(v,j)$ si $i \neq j$

$indice.inf(nouveau(i,j)) = i$

$indice.inf(changer\text{-}ieme(v,i,e)) = indice.inf(v)$

$indice.sup(nouveau(i,j)) = j$

$indice.sup(changer\text{-}ieme(v,i,e)) = indice.sup(v)$

Énoncé

Écrire la spécification algébrique du type $Ensemble[T]$.

Précisions :

- T est le type générique que contient un *Ensemble*
- Les propriétés du type *Ensemble* sont :
  - vide : le constructeur
  - cardinal : nombre d'élément de l'ensemble
  - appartient : test si un élément appartient à l'ensemble
  - ajouter : ajoute un élément à l'ensemble
  - union : réalise l'union de deux ensembles
  - intersection : réalise l'intersection de deux ensembles

Solution

Observateurs	Opérations internes
$cardinal : E[T] \rightarrow I$	$vide : \rightarrow E[T]$
$appartient : E[T] \times T \rightarrow Bool$	$ajouter : E[T] \times T \rightarrow E[T]$
	$union : E[T] \times E[T] \rightarrow E[T]$
	$intersection : E[T] \times E[T] \rightarrow E[T]$

Solution

Observateurs	Opérations internes
$cardinal : E[T] \to I$	$vide : \to E[T]$
$appartient : E[T] \times T \to Bool$	$ajouter : E[T] \times T \to E[T]$
	$union : E[T] \times E[T] \to E[T]$
	$intersection : E[T] \times E[T] \to E[T]$

Axiomes :

- $card(vide()) =$

Solution

Observateurs	Opérations internes
$cardinal : E[T] \rightarrow I$	$vide : \rightarrow E[T]$
$appartient : E[T] \times T \rightarrow Bool$	$ajouter : E[T] \times T \rightarrow E[T]$
	$union : E[T] \times E[T] \rightarrow E[T]$
	$intersection : E[T] \times E[T] \rightarrow E[T]$

Axiomes :

- $card(vide()) = 0$

Solution

Observateurs	Opérations internes
$cardinal : E[T] \rightarrow I$	$vide : \rightarrow E[T]$
$appartient : E[T] \times T \rightarrow Bool$	$ajouter : E[T] \times T \rightarrow E[T]$
	$union : E[T] \times E[T] \rightarrow E[T]$
	$intersection : E[T] \times E[T] \rightarrow E[T]$

Axiomes :

- $card(vide()) = 0$
- $card(ajouter(ens, el)) =$

Solution

Observateurs	Opérations internes
$cardinal : E[T] \rightarrow I$	$vide : \rightarrow E[T]$
$appartient : E[T] \times T \rightarrow Bool$	$ajouter : E[T] \times T \rightarrow E[T]$
	$union : E[T] \times E[T] \rightarrow E[T]$
	$intersection : E[T] \times E[T] \rightarrow E[T]$

Axiomes :

- $card(vide()) = 0$
- $card(ajouter(ens, el)) = cardinal(ens) + 1$

Solution

Observateurs	Opérations internes
$cardinal : E[T] \rightarrow I$	$vide : \rightarrow E[T]$
$appartient : E[T] \times T \rightarrow Bool$	$ajouter : E[T] \times T \rightarrow E[T]$
	$union : E[T] \times E[T] \rightarrow E[T]$
	$intersection : E[T] \times E[T] \rightarrow E[T]$

Axiomes :

- $card(vide()) = 0$
- $card(ajouter(ens, el)) = cardinal(ens) + 1$
- $card(union(ens_1, vide())) =$
- $card(union(ens_1, ens_2)) =$

Solution

Observateurs	Opérations internes
$cardinal : E[T] \to I$	$vide :\to E[T]$
$appartient : E[T] \times T \to Bool$	$ajouter : E[T] \times T \to E[T]$
	$union : E[T] \times E[T] \to E[T]$
	$intersection : E[T] \times E[T] \to E[T]$

Axiomes :

- $card(vide()) = 0$
- $card(ajouter(ens, el)) = cardinal(ens) + 1$
- $card(union(ens_1, vide())) = card(ens_1)$
- $card(union(ens_1, ens_2)) =$
  $card(ens_1) + card(ens_2) - card(intersection(ens_1, ens_2))$

Solution

Observateurs	Opérations internes
$cardinal : E[T] \rightarrow I$	$vide : \rightarrow E[T]$
$appartient : E[T] \times T \rightarrow Bool$	$ajouter : E[T] \times T \rightarrow E[T]$
	$union : E[T] \times E[T] \rightarrow E[T]$
	$intersection : E[T] \times E[T] \rightarrow E[T]$

Axiomes :

- $card(vide()) = 0$
- $card(ajouter(ens, el)) = cardinal(ens) + 1$
- $card(union(ens_1, vide())) = card(ens_1)$
- $card(union(ens_1, ens_2)) =$
  $card(ens_1) + card(ens_2) - card(intersection(ens_1, ens_2))$
- $card(intersection(ens_1, vide())) =$
- $card(intersection(ens_1, ens_2)) =$

Solution

Observateurs	Opérations internes
$cardinal : E[T] \to I$	$vide : \to E[T]$
$appartient : E[T] \times T \to Bool$	$ajouter : E[T] \times T \to E[T]$
	$union : E[T] \times E[T] \to E[T]$
	$intersection : E[T] \times E[T] \to E[T]$

Axiomes :

- $card(vide()) = 0$
- $card(ajouter(ens, el)) = cardinal(ens) + 1$
- $card(union(ens_1, vide())) = card(ens_1)$
- $card(union(ens_1, ens_2)) =$
  $card(ens_1) + card(ens_2) - card(intersection(ens_1, ens_2))$
- $card(intersection(ens_1, vide())) = 0$
- $card(intersection(ens_1, ens_2)) =$
  $card(ens_1) + cardinal(ens_2) - card(union(ens_1, ens_2))$

Solution

Observateurs	Opérations internes
$cardinal : E[T] \rightarrow I$	$vide : \rightarrow E[T]$
$appartient : E[T] \times T \rightarrow Bool$	$ajouter : E[T] \times T \rightarrow E[T]$
	$union : E[T] \times E[T] \rightarrow E[T]$
	$intersection : E[T] \times E[T] \rightarrow E[T]$

Axiomes :

- $appartient(vide(), el) =$

Solution

Observateurs	Opérations internes
$cardinal : E[T] \rightarrow I$	$vide : \rightarrow E[T]$
$appartient : E[T] \times T \rightarrow Bool$	$ajouter : E[T] \times T \rightarrow E[T]$
	$union : E[T] \times E[T] \rightarrow E[T]$
	$intersection : E[T] \times E[T] \rightarrow E[T]$

Axiomes :

- $appartient(vide(), el) = faux$

Solution

Observateurs	Opérations internes
$cardinal : E[T] \to I$	$vide : \to E[T]$
$appartient : E[T] \times T \to Bool$	$ajouter : E[T] \times T \to E[T]$
	$union : E[T] \times E[T] \to E[T]$
	$intersection : E[T] \times E[T] \to E[T]$

Axiomes :

- $appartient(vide(), el) = faux$
- $appartient(ajouter(ens, el_1), el_2) =$

Solution

Observateurs	Opérations internes
$cardinal : E[T] \to I$	$vide : \to E[T]$
$appartient : E[T] \times T \to Bool$	$ajouter : E[T] \times T \to E[T]$
	$union : E[T] \times E[T] \to E[T]$
	$intersection : E[T] \times E[T] \to E[T]$

Axiomes :

- $appartient(vide(), el) = faux$
- $appartient(ajouter(ens, el_1), el_2) =$
  vrai si $el_1 = el_2$, $appartient(ens, el2)$ si $el_1 \neq el_2$

Solution

Observateurs	Opérations internes
$cardinal : E[T] \rightarrow I$	$vide : \rightarrow E[T]$
$appartient : E[T] \times T \rightarrow Bool$	$ajouter : E[T] \times T \rightarrow E[T]$
	$union : E[T] \times E[T] \rightarrow E[T]$
	$intersection : E[T] \times E[T] \rightarrow E[T]$

Axiomes :

- $appartient(vide(), el) = faux$
- $appartient(ajouter(ens, el_1), el_2) =$
  vrai si $el_1 = el_2$, $appartient(ens, el2)$ si $el_1 \neq el_2$
- $appartient(union(ens_1, vide()), el) =$
- $appartient(union(ens_1, ens_2), el) =$

Solution

Observateurs	Opérations internes
$cardinal : E[T] \rightarrow I$	$vide : \rightarrow E[T]$
$appartient : E[T] \times T \rightarrow Bool$	$ajouter : E[T] \times T \rightarrow E[T]$
	$union : E[T] \times E[T] \rightarrow E[T]$
	$intersection : E[T] \times E[T] \rightarrow E[T]$

Axiomes :

- $appartient(vide(), el) = faux$
- $appartient(ajouter(ens, el_1), el_2) =$
  vrai si $el_1 = el_2$, $appartient(ens, el2)$ si $el_1 \neq el_2$
- $appartient(union(ens_1, vide()), el) = appartient(ens_1, el)$
- $appartient(union(ens_1, ens_2), el) =$
  $appartient(ens_1, el) \,||\, appartien(ens_2, el)$

Solution

Observateurs	Opérations internes
$cardinal : E[T] \to I$	$vide : \to E[T]$
$appartient : E[T] \times T \to Bool$	$ajouter : E[T] \times T \to E[T]$
	$union : E[T] \times E[T] \to E[T]$
	$intersection : E[T] \times E[T] \to E[T]$

Axiomes :

- $appartient(vide(), el) = faux$
- $appartient(ajouter(ens, el_1), el_2) =$
  $\text{vrai si } el_1 = el_2, appartient(ens, el2) \text{ si } el_1 \neq el_2$
- $appartient(union(ens_1, vide()), el) = appartient(ens_1, el)$
- $appartient(union(ens_1, ens_2), el) =$
  $appartient(ens_1, el) \mid\mid appartien(ens_2, el)$
- $appartient(intersection(ens_1, vide()), el) =$
- $appartient(intersection(ens_1, ens_2), el) =$

Solution

Observateurs	Opérations internes
$cardinal : E[T] \rightarrow I$	$vide : \rightarrow E[T]$
$appartient : E[T] \times T \rightarrow Bool$	$ajouter : E[T] \times T \rightarrow E[T]$
	$union : E[T] \times E[T] \rightarrow E[T]$
	$intersection : E[T] \times E[T] \rightarrow E[T]$

Axiomes :

- $appartient(vide(), el) = faux$
- $appartient(ajouter(ens, el_1), el_2) =$
  vrai si $el_1 = el_2$, $appartient(ens, el2)$ si $el_1 \neq el_2$
- $appartient(union(ens_1, vide()), el) = appartient(ens_1, el)$
- $appartient(union(ens_1, ens_2), el) =$
  $appartient(ens_1, el) \,||\, appartien(ens_2, el)$
- $appartient(intersection(ens_1, vide()), el) = faux$
- $appartient(intersection(ens_1, ens_2), el) =$
  $appartient(ens_1, el) \,\&\&\, appartien(ens_2, el)$

Plan

Specification vs. Implantation

Type abstrait	Classe (*Java*)
Opérations	
Observateurs	
$ieme : V \times I \to E$	
Opérations internes	
$changer\text{-}ieme : V \times I \times E \to V$	

Specification vs. Implantation

Type abstrait	Classe (*Java*)
Opérations	Fonctions
Observateurs	
$ieme : V \times I \rightarrow E$	
Opérations internes	
$changer\text{-}ieme : V \times I \times E \rightarrow V$	

Specification vs. Implantation

Type abstrait	Classe (*Java*)
Opérations	Fonctions
Observateurs	Getters
$ieme : V \times I \to E$	
Opérations internes	
$changer\text{-}ieme : V \times I \times E \to V$	

Specification vs. Implantation

Type abstrait	Classe (*Java*)
Opérations	Fonctions
Observateurs	Getters
$ieme : V \times I \to E$	*E ieme(I i);*
Opérations internes	
$changer\text{-}ieme : V \times I \times E \to V$	

Specification vs. Implantation

Type abstrait	Classe (*Java*)
Opérations	Fonctions
Observateurs	Getters
$ieme : V \times I \to E$	*E ieme(I i);*
Opérations internes	Setters
$changer\text{-}ieme : V \times I \times E \to V$	

Specification vs. Implantation

Type abstrait	Classe (*Java*)
Opérations	Fonctions
Observateurs	Getters
$ieme : V \times I \to E$	*E ieme(I i);*
Opérations internes	Setters
$changer\text{-}ieme : V \times I \times E \to V$	*void changer-ieme(I i, E e);*

Implantations multiples

Un **type abstrait** peut être **implanté** dans n'importe quel **langage de programmation** à partir de sa **spécification algébrique**.

Exemple

Le type Vecteur est implanté :

- dans la classe Java *Vector*<*T*> *(Doc)*
- dans la classe C++ *std* :: *vector* *(Doc)*

Exercice 1.B : Énoncé

Écrire la classe générique Triplet avec :

- Un constructeur à trois arguments
- Trois méthodes d'accès *getPremier*, *getSecond* et *getTroisieme*
- Une méthode *affiche* affichant le triplet

Exercice 1.B : Correction

```
class Triplet<T>
{
    private T x, y, z;   // Les trois éléments du triplet

    public Triplet( T premier, T second, T troisieme )
    {
        x = premier; y = second; z = troisieme;
    }

    public T getPremier() { return x; }
    public T getSecond() { return y; }
    public T getTroisieme() { return z; }

    public void affiche()
    {
      System.out.println("Triplet:("+x+", "+y+", "+z+")");
    }

}
```

Exercice 1.C

Repérez les erreurs dans le code ci-dessous :

```java
public class C<T>
{
    T x ;
    T[] t1 ;
    T[] t2 ;
    static T inf ;
    static int compte ;
    void f()
    {
        x = new T();
        t2 = t1;
        t2 = new T[5];
    }
}
```

Exercice 1.C

Repérez les erreurs dans le code ci-dessous :

```java
public class C<T>
{
    T x ;                     // Ok
    T[] t1 ;
    T[] t2 ;
    static T inf ;
    static int compte ;
    void f()
    {
        x = new T();
        t2 = t1;
        t2 = new T[5];
    }
}
```

Exercice 1.C

Repérez les erreurs dans le code ci-dessous :

```
public class C<T>
{
    T x ;                       // Ok
    T[] t1 ;                    // Ok
    T[] t2 ;
    static T inf ;
    static int compte ;
    void f()
    {
        x = new T();
        t2 = t1;
        t2 = new T[5];
    }
}
```

Exercice 1.C

Repérez les erreurs dans le code ci-dessous :

```
public class C<T>
{
    T x ;                    // Ok
    T[] t1 ;                 // Ok
    T[] t2 ;                 // Ok
    static T inf ;
    static int compte ;
    void f()
    {
        x = new T();
        t2 = t1;
        t2 = new T[5];
    }
}
```

Exercice 1.C

Repérez les erreurs dans le code ci-dessous :

```
public class C<T>
{
    T x ;                      // Ok
    T[] t1 ;                   // Ok
    T[] t2 ;                   // Ok
    static T inf ;             // Interdit !
    static int compte ;
    void f()
    {
        x = new T();
        t2 = t1;
        t2 = new T[5];
    }
}
```

Exercice 1.C

Repérez les erreurs dans le code ci-dessous :

```
public class C<T>
{
    T x ;                   // Ok
    T[] t1 ;                // Ok
    T[] t2 ;                // Ok
    static T inf ;          // Interdit !
    static int compte ;     // Ok
    void f()
    {
        x = new T();
        t2 = t1;
        t2 = new T[5];
    }
}
```

Exercice 1.C

Repérez les erreurs dans le code ci-dessous :

```
public class C<T>
{
    T x ;                       // Ok
    T[] t1 ;                    // Ok
    T[] t2 ;                    // Ok
    static T inf ;              // Interdit !
    static int compte ;         // Ok
    void f()
    {
        x = new T();            // Interdit !
        t2 = t1;
        t2 = new T[5];
    }
}
```

Exercice 1.C

Repérez les erreurs dans le code ci-dessous :

```
public class C<T>
{
    T x ;                   // Ok
    T[] t1 ;                // Ok
    T[] t2 ;                // Ok
    static T inf ;          // Interdit !
    static int compte ;     // Ok
    void f()
    {
        x = new T();        // Interdit !
        t2 = t1;            // Ok
        t2 = new T[5];
    }
}
```

Exercice 1.C

Repérez les erreurs dans le code ci-dessous :

```
public class C<T>
{
    T x ;                   // Ok
    T[] t1 ;                // Ok
    T[] t2 ;                // Ok
    static T inf ;          // Interdit !
    static int compte ;     // Ok
    void f()
    {
        x = new T();        // Interdit !
        t2 = t1;            // Ok
        t2 = new T[5];      // Interdit !
    }
}
```

Plan

Boites noires et boites blanches

Deux méthodes de tests :

Boite noire on ne connaît pas l'implantation
$\rightarrow$ tests du comportement des fonctions

Boite blanche on connaît l'implantation
$\rightarrow$ tests de chaque implantation

Boites noires et boites blanches

Deux méthodes de tests :

Boite noire on ne connaît pas l'implantation
$\rightarrow$ tests du comportement des fonctions

Boite blanche on connaît l'implantation
$\rightarrow$ tests de chaque implantation

Nous ne verrons que les tests par boite noire :
ils s'appuient sur la spécification algébrique.

Les 3 points essentiels pour écrire de bons tests

Les 3 points essentiels pour écrire de bons tests

Généricité Écrire des tests valables pour toutes les implantations.

Les 3 points essentiels pour écrire de bons tests

Généricité Écrire des tests valables pour toutes les implantations.

Couverture Tester l'ensemble des comportements des classes.

Les 3 points essentiels pour écrire de bons tests

Généricité Écrire des tests valables pour toutes les implantations.

Couverture Tester l'ensemble des comportements des classes.

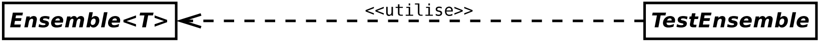
Clarté Les résultats des tests doivent être faciles à comprendre.

Mutualisation et hiérarchie

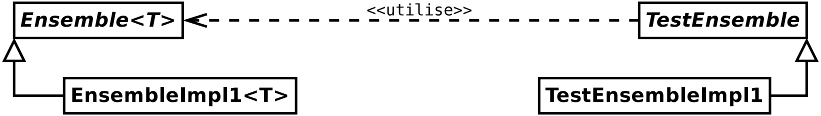
Mutualisation et hiérarchie

Ensemble<T>

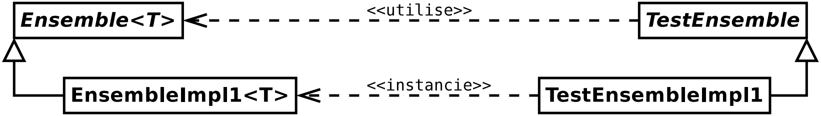
Mutualisation et hiérarchie



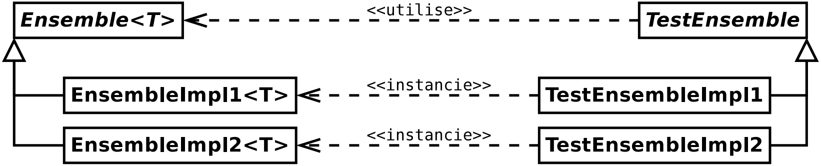
Mutualisation et hiérarchie



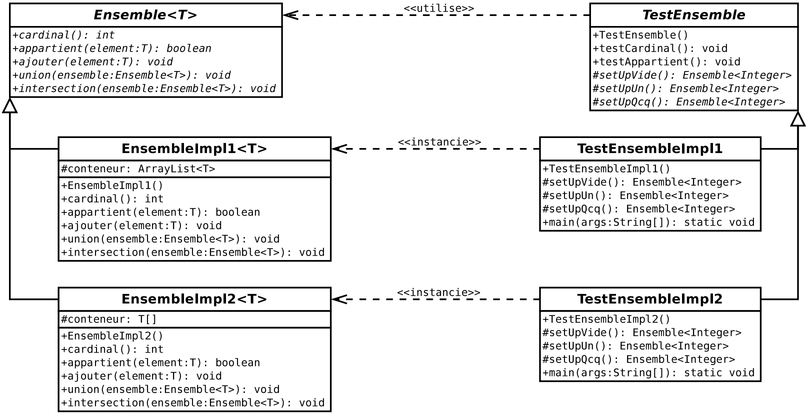
Mutualisation et hiérarchie



Mutualisation et hiérarchie



Contenu des classes



Morceaux choisis

```
abstract class TestEnsemble
{
    public TestEnsemble()
    {
        testCardinal();
        testAppartient();
    }

    private void testCardinal() { ... }
    private void testAppartient() { ... }

    protected abstract Ensemble<Integer> setupVide();
    protected abstract Ensemble<Integer> setupUn();
    protected abstract Ensemble<Integer> setupQcq();
}
```

Morceaux choisis

```java
class TestEnsembleImpl1 extends TestEnsemble
{
    public TestEnsembleImpl1() { super(); }

    protected Ensemble<Integer> setupVide()
    {
        return new EnsembleImpl1<Integer>();
    }

    protected Ensemble<Integer> setupUn() {...}
    protected Ensemble<Integer> setupQcq() {...}

    public static void main( String[] args )
    {
        new TestEnsembleImpl1();
    }
}
```

Morceaux choisis

```
abstract class TestEnsemble
{
    public void testCardinal()
    {
        // cardinal(vide()) = 0
          ...
        // cardinal(ajouter(ens,el)) = cardinal(ens) + 1
          ...
    }
}
```

Morceaux choisis

```
abstract class TestEnsemble
{
    public void testCardinal()
    {
        ...
        // gauche = droit
        1. Construction de droit
        2. Construction de gauche
        3. Si gauche est différent de droit, exception
        ...
    }
}
```

Morceaux choisis

```
abstract class TestEnsemble
{
    public void testCardinal()
    {
        // cardinal(vide()) = 0
        Ensemble<Integer> ens = setupVide ();
        int gauche = ens.cardinal();
        assert(gauche == 0) : "Bug cardinal(vide()) = 0";
        // cardinal(ajouter(ens,el)) = cardinal(ens) + 1
        ...
    }
}
```

Morceaux choisis

```
abstract class TestEnsemble
{
    public void testCardinal()
    {
        // cardinal(vide()) = 0
        ...
        // cardinal(ajouter(ens,e)) = cardinal(ens) + 1
        //  1. cas où ens est vide
        //  2. cas où ens est réduit à un élément
        //  3. cas où ens contient plus d'un élément
        ...
    }
}
```

Morceaux choisis

```java
abstract class TestEnsemble
{
  public void testCardinal()
  {
    // cardinal(ajouter(ens,e)) = cardinal(ens) + 1
    //  1. cas où ens est vide
    Ensemble<Integer> ens = setupVide() ;
    Integer e ;
    do {
        e = new Integer ((int)(Math.random( )*100)) ;
    } while (ens.appartient(e)); // precondition
    ens.ajouter(e);
    int gauche = ens.cardinal();
    assert (gauche == 1) :
    "bug ajouter(ens,e))=cardinal(ens)+1 avec ens vide";
    //  2. cas où ens est réduit à un élément
    //  3. cas où ens contient plus d'un élément
    ...
  }
}
```

Morceaux choisis

```java
abstract class TestEnsemble
{
  public void testCardinal()
  {
    // cardinal(ajouter(ens,e)) = cardinal(ens) + 1
    //  1. cas où ens est vide
    //  2. cas où ens est réduit à un élément
    //  3. cas où ens contient plus d'un élément
    Ensemble<Integer> ens = setupQcq() ;
    int droit = ens.cardinal() + 1 ;
    Integer e ;
    do {
        e = new Integer ((int)(Math.random( )*100)) ;
    } while (ens.appartient(e)); // precondition
    ens.ajouter(e);
    int gauche = ens.cardinal();
    assert (gauche == droit) :
    "bug ajouter(ens,e))=cardinal(ens)+1 avec ens vide";
  }
}
```

Morceaux choisis

```
abstract class TestEnsemble
{
    public void testCardinal()
    {
        // cardinal(vide()) = 0
        testCardinalVide();
        // cardinal(ajouter(ens,e)) = cardinal(ens) + 1
        testCardinalAjouter();
    }

    public void testCardinalVide() { ... }

    public void testCardinalAjouter()
    {
        //  1. cas où ens est vide
        //  2. cas où ens est réduit à un élément
        //  3. cas où ens contient plus d'un élément
    }
}
```

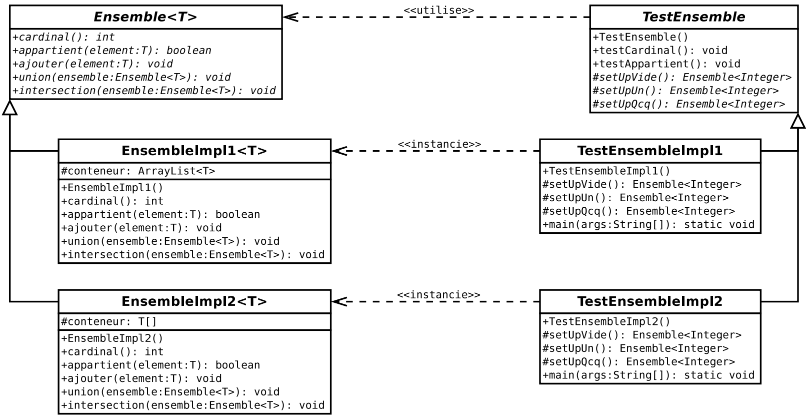
Morceaux choisis

```java
abstract class TestEnsemble
{
    public void testCardinal()
    {
        // cardinal(ajouter(ens,e)) = cardinal(ens) + 1
        testCardinalAjouterEnsE();
    }

    public void testCardinalAjouter()
    {
        testCardinalAjouterVide();
        testCardinalAjouterUn();
        testCardinalAjouterQcq();
    }

    public void testCardinalAjouterVide() { ... }
    public void testCardinalAjouterUn() { ... }
    public void testCardinalAjouterQcq() { ... }
}
```

Morceaux choisis

```
class TestEnsembleImpl1 extends TestEnsemble
{
    public TestEnsembleImpl1<Integer> setupVide()
    {
        return new EnsembleImpl1<Integer>();
    }

    public TestEnsembleImpl1<Integer> setupUn()
    {
        EnsembleImpl1<Integer> ens;
        ens = new EnsembleImpl1<Integer>();
        ens.ajouter( new Integer(5) );
        return ens;
    }
}
```

Récapitulatif



Plan

Opérations

$$
\begin{array}{rll}
nouveau: & & \rightarrow Tarif \\
ajouter: & Tarif \times Produit \times Reel & \rightarrow Tarif \\
existe: & Tarif \times Produit & \rightarrow Booleen \\
prix: & Tarif \times Produit & \rightarrow Reel \\
nbProduits: & Tarif & \rightarrow Entier \\
supprimer: & Tarif \times Produit & \rightarrow Tarif \\
modifier: & Tarif \times Produit \times Reel & \rightarrow Tarif \\
fusion: & Tarif \times Tarif & \rightarrow Tarif \\
enCommun: & Tarif \times Tarif & \rightarrow Tarif
\end{array}
$$

Opérations

$$
\begin{array}{rlr}
\textit{nouveau}: & & \rightarrow \textit{Tarif}\\
\textit{ajouter}: & \textit{Tarif} \times \textit{Produit} \times \textit{Reel} & \rightarrow \textit{Tarif}\\
\textit{existe}: & \textit{Tarif} \times \textit{Produit} & \rightarrow \textit{Booleen}\\
\textit{prix}: & \textit{Tarif} \times \textit{Produit} & \rightarrow \textit{Reel}\\
\textit{nbProduits}: & \textit{Tarif} & \rightarrow \textit{Entier}\\
\textit{supprimer}: & \textit{Tarif} \times \textit{Produit} & \rightarrow \textit{Tarif}\\
\textit{modifier}: & \textit{Tarif} \times \textit{Produit} \times \textit{Reel} & \rightarrow \textit{Tarif}\\
\textit{fusion}: & \textit{Tarif} \times \textit{Tarif} & \rightarrow \textit{Tarif}\\
\textit{enCommun}: & \textit{Tarif} \times \textit{Tarif} & \rightarrow \textit{Tarif}
\end{array}
$$

Opérations

$$
\begin{array}{rll}
\textit{nouveau} : & & \rightarrow \textit{Tarif} \\
\textit{ajouter} : & \textit{Tarif} \times \textit{Produit} \times \textit{Reel} & \rightarrow \textit{Tarif} \\
\textit{existe} : & \textit{Tarif} \times \textit{Produit} & \rightarrow \textit{Booleen} \\
\textit{prix} : & \textit{Tarif} \times \textit{Produit} & \rightarrow \textit{Reel} \\
\textit{nbProduits} : & \textit{Tarif} & \rightarrow \textit{Entier} \\
\textit{supprimer} : & \textit{Tarif} \times \textit{Produit} & \rightarrow \textit{Tarif} \\
\textit{modifier} : & \textit{Tarif} \times \textit{Produit} \times \textit{Reel} & \rightarrow \textit{Tarif} \\
\textit{fusion} : & \textit{Tarif} \times \textit{Tarif} & \rightarrow \textit{Tarif} \\
\textit{enCommun} : & \textit{Tarif} \times \textit{Tarif} & \rightarrow \textit{Tarif}
\end{array}
$$

Les 4 préconditions

Les 4 préconditions

$$\mathit{ajouter}(t, \mathit{pro}, \mathit{px}) \quad \text{defini ssi}$$

Les 4 préconditions

$$ajouter(t, pro, px) \quad \text{defini ssi} \quad non\ existe(t, pro)$$

Les 4 préconditions

$$\begin{aligned} ajouter(t, pro, px) \quad & \text{defini ssi} \quad non\ existe(t, pro) \\ prix(t, pro) \quad & \text{defini ssi} \end{aligned}$$

Les 4 préconditions

$$
\begin{aligned}
ajouter(t, pro, px) &\quad \text{defini ssi} \quad non\ existe(t, pro) \\
prix(t, pro) &\quad \text{defini ssi} \quad existe(t, pro)
\end{aligned}
$$

Les 4 préconditions

$$\begin{array}{rll}
ajouter(t, pro, px) & \text{defini ssi} & non\ existe(t, pro) \\
prix(t, pro) & \text{defini ssi} & existe(t, pro) \\
supprimer(t, pro) & \text{defini ssi} &
\end{array}$$

Les 4 préconditions

$$
\begin{aligned}
ajouter(t, pro, px) &\quad \text{defini ssi} \quad non\ existe(t, pro)\\
prix(t, pro) &\quad \text{defini ssi} \quad existe(t, pro)\\
supprimer(t, pro) &\quad \text{defini ssi} \quad existe(t, pro)
\end{aligned}
$$

Les 4 préconditions

$$\begin{array}{rll}
ajouter(t, pro, px) & \text{defini ssi} & non\ existe(t, pro) \\
prix(t, pro) & \text{defini ssi} & existe(t, pro) \\
supprimer(t, pro) & \text{defini ssi} & existe(t, pro) \\
modifier(t, pro, px) & \text{defini ssi} &
\end{array}$$

Les 4 préconditions

$$\begin{array}{rll}
ajouter(t, pro, px) & \text{defini ssi} & non\ existe(t, pro) \\
prix(t, pro) & \text{defini ssi} & existe(t, pro) \\
supprimer(t, pro) & \text{defini ssi} & existe(t, pro) \\
modifier(t, pro, px) & \text{defini ssi} & existe(t, pro)
\end{array}$$

Axiomes avec *existe*

Axiomes avec *existe*

$$existe(nouveau(), pro) \quad =$$

Axiomes avec *existe*

$$existe(nouveau(), pro) \quad = \quad faux$$

Axiomes avec *existe*

$$
\begin{aligned}
existe(nouveau(), pro) &= faux \\
existe(ajouter(t, pro_1, px), pro_2) &=
\end{aligned}
$$

Axiomes avec *existe*

$$
\begin{aligned}
existe(nouveau(), pro) &= faux \\
existe(ajouter(t, pro_1, px), pro_2) &= \begin{cases} vrai & \text{si } pro_1 = pro_2 \\ existe(t, pro_2) & \text{sinon} \end{cases}
\end{aligned}
$$

Axiomes avec *existe*

$$
\begin{aligned}
existe(nouveau(), pro) &= faux \\
existe(ajouter(t, pro_1, px), pro_2) &= \begin{cases} vrai & \text{si } pro_1 = pro_2 \\ existe(t, pro_2) & \text{sinon} \end{cases} \\
existe(supprimer(t, pro_1), pro_2) &=
\end{aligned}
$$

Axiomes avec *existe*

$$
\begin{aligned}
existe(nouveau(), pro) &= faux \\
existe(ajouter(t, pro_1, px), pro_2) &= \begin{cases} vrai & \text{si } pro_1 = pro_2 \\ existe(t, pro_2) & \text{sinon} \end{cases} \\
existe(supprimer(t, pro_1), pro_2) &= \begin{cases} faux & \text{si } pro_1 = pro_2 \\ existe(t, pro_2) & \text{sinon} \end{cases}
\end{aligned}
$$

Axiomes avec *existe*

$$
\begin{aligned}
existe(nouveau(), pro) &= faux \\
existe(ajouter(t, pro_1, px), pro_2) &= \begin{cases} vrai & \text{si } pro_1 = pro_2 \\ existe(t, pro_2) & \text{sinon} \end{cases} \\
existe(supprimer(t, pro_1), pro_2) &= \begin{cases} faux & \text{si } pro_1 = pro_2 \\ existe(t, pro_2) & \text{sinon} \end{cases} \\
existe(modifier(t, pro_1, px), pro_2) &=
\end{aligned}
$$

Axiomes avec *existe*

$$
\begin{aligned}
existe(nouveau(), pro) &= faux \\
existe(ajouter(t, pro_1, px), pro_2) &= \begin{cases} vrai & \text{si } pro_1 = pro_2 \\ existe(t, pro_2) & \text{sinon} \end{cases} \\
existe(supprimer(t, pro_1), pro_2) &= \begin{cases} faux & \text{si } pro_1 = pro_2 \\ existe(t, pro_2) & \text{sinon} \end{cases} \\
existe(modifier(t, pro_1, px), pro_2) &= existe(t, pro_2)
\end{aligned}
$$

Axiomes avec *existe*

$$
\begin{aligned}
existe(nouveau(), pro) &= faux \\
existe(ajouter(t, pro_1, px), pro_2) &= \begin{cases} vrai & \text{si } pro_1 = pro_2 \\ existe(t, pro_2) & \text{sinon} \end{cases} \\
existe(supprimer(t, pro_1), pro_2) &= \begin{cases} faux & \text{si } pro_1 = pro_2 \\ existe(t, pro_2) & \text{sinon} \end{cases} \\
existe(modifier(t, pro_1, px), pro_2) &= existe(t, pro_2) \\
existe(fusion(t_1, t_2), pro) &=
\end{aligned}
$$

Axiomes avec *existe*

$$
\begin{aligned}
existe(nouveau(), pro) &= faux \\
existe(ajouter(t, pro_1, px), pro_2) &= \begin{cases} vrai & \text{si } pro_1 = pro_2 \\ existe(t, pro_2) & \text{sinon} \end{cases} \\
existe(supprimer(t, pro_1), pro_2) &= \begin{cases} faux & \text{si } pro_1 = pro_2 \\ existe(t, pro_2) & \text{sinon} \end{cases} \\
existe(modifier(t, pro_1, px), pro_2) &= existe(t, pro_2) \\
existe(fusion(t_1, t_2), pro) &= existe(t_1, pro) || existe(t_2, pro)
\end{aligned}
$$

Axiomes avec *existe*

$$
\begin{aligned}
existe(nouveau(), pro) &= faux \\
existe(ajouter(t, pro_1, px), pro_2) &= \begin{cases} vrai & \text{si } pro_1 = pro_2 \\ existe(t, pro_2) & \text{sinon} \end{cases} \\
existe(supprimer(t, pro_1), pro_2) &= \begin{cases} faux & \text{si } pro_1 = pro_2 \\ existe(t, pro_2) & \text{sinon} \end{cases} \\
existe(modifier(t, pro_1, px), pro_2) &= existe(t, pro_2) \\
existe(fusion(t_1, t_2), pro) &= existe(t_1, pro) || existe(t_2, pro) \\
existe(enCommun(t_1, t_2), pro) &=
\end{aligned}
$$

Axiomes avec *existe*

$$
\begin{aligned}
existe(nouveau(), pro) &= faux \\
existe(ajouter(t, pro_1, px), pro_2) &= \begin{cases} vrai & \text{si } pro_1 = pro_2 \\ existe(t, pro_2) & \text{sinon} \end{cases} \\
existe(supprimer(t, pro_1), pro_2) &= \begin{cases} faux & \text{si } pro_1 = pro_2 \\ existe(t, pro_2) & \text{sinon} \end{cases} \\
existe(modifier(t, pro_1, px), pro_2) &= existe(t, pro_2) \\
existe(fusion(t_1, t_2), pro) &= existe(t_1, pro) || existe(t_2, pro) \\
existe(enCommun(t_1, t_2), pro) &= existe(t_1, pro) \&\& existe(t_2, pro)
\end{aligned}
$$

Axiomes avec *prix*

Axiomes avec *prix*

$$prix(nouveau(), pro) \quad =$$

Axiomes avec *prix*

$$prix(nouveau(), pro) \quad = \quad \text{Violation de précondition !}$$

Axiomes avec *prix*

$$
\begin{aligned}
prix(nouveau(), pro) &= \text{Violation de précondition !}\\
prix(ajouter(t, pro_1, px), pro_2) &=
\end{aligned}
$$

Axiomes avec *prix*

$$prix(nouveau(), pro) = \text{Violation de précondition !}$$

$$prix(ajouter(t, pro_1, px), pro_2) = \begin{cases} px & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases}$$

Axiomes avec *prix*

$$
\begin{aligned}
prix(nouveau(), pro) &= \text{Violation de précondition !}\\
prix(ajouter(t, pro_1, px), pro_2) &= \begin{cases} px & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases}\\
prix(supprimer(t, pro_1), pro_2) &=
\end{aligned}
$$

Axiomes avec *prix*

$$\begin{aligned}
prix(nouveau(), pro) &= \text{Violation de précondition !}\\
prix(ajouter(t, pro_1, px), pro_2) &= \begin{cases} px & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases}\\
prix(supprimer(t, pro_1), pro_2) &= \begin{cases} Interdit! & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases}
\end{aligned}$$

Axiomes avec *prix*

$$
\begin{aligned}
prix(nouveau(), pro) &= \text{Violation de précondition !} \\
prix(ajouter(t, pro_1, px), pro_2) &= \begin{cases} px & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases} \\
prix(supprimer(t, pro_1), pro_2) &= \begin{cases} \textit{Interdit}! & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases} \\
prix(modifier(t, pro_1, px), pro_2) &=
\end{aligned}
$$

Axiomes avec *prix*

$$\begin{aligned}
prix(nouveau(), pro) &= \text{Violation de précondition !}\\
prix(ajouter(t, pro_1, px), pro_2) &= \begin{cases} px & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases}\\
prix(supprimer(t, pro_1), pro_2) &= \begin{cases} \textit{Interdit!} & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases}\\
prix(modifier(t, pro_1, px), pro_2) &= \begin{cases} px & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases}
\end{aligned}$$

Axiomes avec *prix*

$$
\begin{aligned}
prix(nouveau(), pro) &= \text{Violation de précondition !}\\
prix(ajouter(t, pro_1, px), pro_2) &= \begin{cases} px & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases}\\
prix(supprimer(t, pro_1), pro_2) &= \begin{cases} \mathit{Interdit!} & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases}\\
prix(modifier(t, pro_1, px), pro_2) &= \begin{cases} px & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases}\\
prix(fusion(t_1, t_2), pro) &=
\end{aligned}
$$

Axiomes avec *prix*

$$
\begin{aligned}
prix(nouveau(), pro) &= \text{Violation de précondition !} \\
prix(ajouter(t, pro_1, px), pro_2) &= \begin{cases} px & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases} \\
prix(supprimer(t, pro_1), pro_2) &= \begin{cases} \textit{Interdit}! & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases} \\
prix(modifier(t, pro_1, px), pro_2) &= \begin{cases} px & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases} \\
prix(fusion(t_1, t_2), pro) &= min(prix(t_1, pro), prix(t_2, pro))
\end{aligned}
$$

Axiomes avec *prix*

$$
\begin{aligned}
prix(nouveau(), pro) &= \text{Violation de précondition !}\\
prix(ajouter(t, pro_1, px), pro_2) &= \begin{cases} px & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases}\\
prix(supprimer(t, pro_1), pro_2) &= \begin{cases} \textit{Interdit}! & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases}\\
prix(modifier(t, pro_1, px), pro_2) &= \begin{cases} px & \text{si } pro_1 = pro_2 \\ prix(t, pro_2) & \text{sinon} \end{cases}\\
prix(fusion(t_1, t_2), pro) &= min(prix(t_1, pro), prix(t_2, pro))\\
prix(enCommun(t_1, t_2), pro) &=
\end{aligned}
$$

Axiomes avec *prix*

$$
\begin{aligned}
prix(nouveau(), pro) &= \text{Violation de précondition !}\\
prix(ajouter(t, pro_1, px), pro_2) &= \begin{cases} px & \text{si } pro_1 = pro_2\\ prix(t, pro_2) & \text{sinon}\end{cases}\\
prix(supprimer(t, pro_1), pro_2) &= \begin{cases} \textit{Interdit}! & \text{si } pro_1 = pro_2\\ prix(t, pro_2) & \text{sinon}\end{cases}\\
prix(modifier(t, pro_1, px), pro_2) &= \begin{cases} px & \text{si } pro_1 = pro_2\\ prix(t, pro_2) & \text{sinon}\end{cases}\\
prix(fusion(t_1, t_2), pro) &= min(prix(t_1, pro), prix(t_2, pro))\\
prix(enCommun(t_1, t_2), pro) &= prix(t_1, pro)
\end{aligned}
$$

Axiomes avec *nbProduits*

Axiomes avec *nbProduits*

$$nbProduits(nouveau()) \quad =$$

Axiomes avec *nbProduits*

$$nbProduits(nouveau()) \quad = \quad 0$$

Axiomes avec *nbProduits*

$$
\begin{aligned}
nbProduits(nouveau()) &= 0 \\
nbProduits(ajouter(t, pro, px)) &=
\end{aligned}
$$

Axiomes avec *nbProduits*

$$
\begin{array}{rcl}
nbProduits(nouveau()) & = & 0 \\
nbProduits(ajouter(t, pro, px)) & = & nbProduits(t) + 1
\end{array}
$$

Axiomes avec *nbProduits*

$$
\begin{aligned}
nbProduits(nouveau()) &= 0\\
nbProduits(ajouter(t, pro, px)) &= nbProduits(t) + 1\\
nbProduits(supprimer(t, pro)) &=
\end{aligned}
$$

Axiomes avec *nbProduits*

$$
\begin{aligned}
nbProduits(nouveau()) &= 0 \\
nbProduits(ajouter(t, pro, px)) &= nbProduits(t) + 1 \\
nbProduits(supprimer(t, pro)) &= nbProduits(t) - 1
\end{aligned}
$$

Axiomes avec *nbProduits*

$$
\begin{aligned}
nbProduits(nouveau()) &= 0\\
nbProduits(ajouter(t, pro, px)) &= nbProduits(t) + 1\\
nbProduits(supprimer(t, pro)) &= nbProduits(t) - 1\\
nbProduits(modifier(t, pro, px)) &=
\end{aligned}
$$

Axiomes avec *nbProduits*

$$
\begin{array}{rcl}
nbProduits(nouveau()) & = & 0 \\
nbProduits(ajouter(t, pro, px)) & = & nbProduits(t) + 1 \\
nbProduits(supprimer(t, pro)) & = & nbProduits(t) - 1 \\
nbProduits(modifier(t, pro, px)) & = & nbProduits(t)
\end{array}
$$

Axiomes avec *nbProduits*

$$
\begin{array}{rcl}
nbProduits(nouveau()) & = & 0 \\
nbProduits(ajouter(t, pro, px)) & = & nbProduits(t) + 1 \\
nbProduits(supprimer(t, pro)) & = & nbProduits(t) - 1 \\
nbProduits(modifier(t, pro, px)) & = & nbProduits(t) \\
nbProduits(fusion(t_1, t_2)) & = &
\end{array}
$$

Axiomes avec *nbProduits*

$$
\begin{array}{rcl}
nbProduits(nouveau()) & = & 0 \\
nbProduits(ajouter(t, pro, px)) & = & nbProduits(t) + 1 \\
nbProduits(supprimer(t, pro)) & = & nbProduits(t) - 1 \\
nbProduits(modifier(t, pro, px)) & = & nbProduits(t) \\
nbProduits(fusion(t_1, t_2)) & = & nbProduits(t_1) + nbProduits(t_2) \\
 & & -nbProduits(enCommun(t_1, t_2))
\end{array}
$$

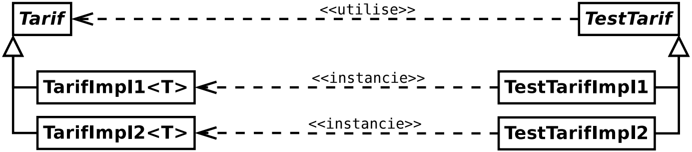
Axiomes avec *nbProduits*

$$
\begin{array}{rcl}
nbProduits(nouveau()) & = & 0 \\
nbProduits(ajouter(t, pro, px)) & = & nbProduits(t) + 1 \\
nbProduits(supprimer(t, pro)) & = & nbProduits(t) - 1 \\
nbProduits(modifier(t, pro, px)) & = & nbProduits(t) \\
nbProduits(fusion(t_1, t_2)) & = & nbProduits(t_1) + nbProduits(t_2) \\
 & & -nbProduits(enCommun(t_1, t_2)) \\
nbProduits(enCommun(t_1, t_2)) & = &
\end{array}
$$

Axiomes avec *nbProduits*

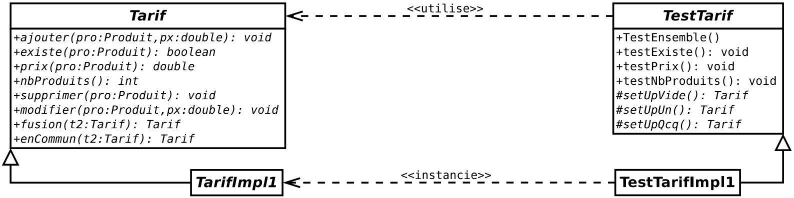
$$
\begin{array}{rcl}
nbProduits(nouveau()) & = & 0 \\
nbProduits(ajouter(t, pro, px)) & = & nbProduits(t) + 1 \\
nbProduits(supprimer(t, pro)) & = & nbProduits(t) - 1 \\
nbProduits(modifier(t, pro, px)) & = & nbProduits(t) \\
nbProduits(fusion(t_1, t_2)) & = & nbProduits(t_1) + nbProduits(t_2) \\
 & & -nbProduits(enCommun(t_1, t_2)) \\
nbProduits(enCommun(t_1, t_2)) & = & nbProduits(t_1) + nbProduits(t_2) \\
 & & -nbProduits(fusion(t_1, t_2))
\end{array}
$$

La classe `TestTarif`



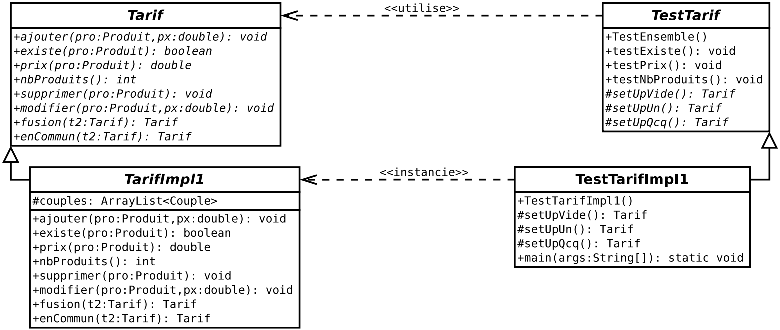
Écrivez la classe `TestTarif`, classe abstraite qui sera valable pour n'importe quelle implantation respectant la spécification algébrique.

La classe `TestTarif`



Écrivez la classe `TestTarif`, classe abstraite qui sera valable pour n'importe quelle implantation respectant la spécification algébrique.

La classe TestTarif



Écrivez la classe `TestTarif`, classe abstraite qui sera valable pour n'importe quelle implantation respectant la spécification algébrique.

Class Tarif

```
package tarif ;
public interface Tarif {

    public void ajouter(Produit prod, double px) ;
    public void modifier(Produit prod, double px) ;
    public void supprimer(Produit prod) ;

    public int nbProduits() ;
    public boolean existe(Produit prod) ;
    public double prix(Produit prod) ;

}
```