Généricité

III. Généricité contrainte
1. Contrainte générique
2. Bornes multiples
IV. Joker
1. Définition
2. Bornes du joker
3. Types à joker borné
4. Sous-typage
5. Instanciation des types paramétrés
V. Méthodes génériques
1. Motivation
2. Définition
3. Inférence des paramètres de types
i. Algorithme d'inférence
ii. Exemple 1
iii. Exemple 2
iv. Exemple 3
4. Capture du joker
i. Motivation et définition
ii. Exemples
5. Règles méthodologiques
i. Méthode générique ou à joker ?
ii. Utilisation des jokers bornés
a. Règles 1 & 2
b. Règles 3 & 4
VI. Compléments
1. Tableaux et généricité
2. Exceptions et généricité
3. Classe Class<E>
4. Types énumérés
i. Classe Enum<E>
ii. Entête de la classe Enum
iii. Exemple élaboré

Contrainte générique à bornes multiples :

`G<E extends [X &] I1 & ... & In>`

- `|E|` = `|X|` (ou `|I1|` si `X` absent)
- une classe, ou un paramètre de type
- interfaces telles que ∀ i,j `|Ii|` ≠ `|Ij|`

~~~~
class G<E extends Comparable<Human> & Comparable<Student>> {
    ...
}
~~~~

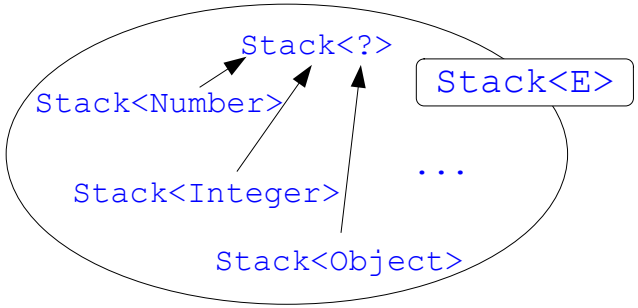
```
class G<E extends Number & CharSequence> {
    ...
}
```

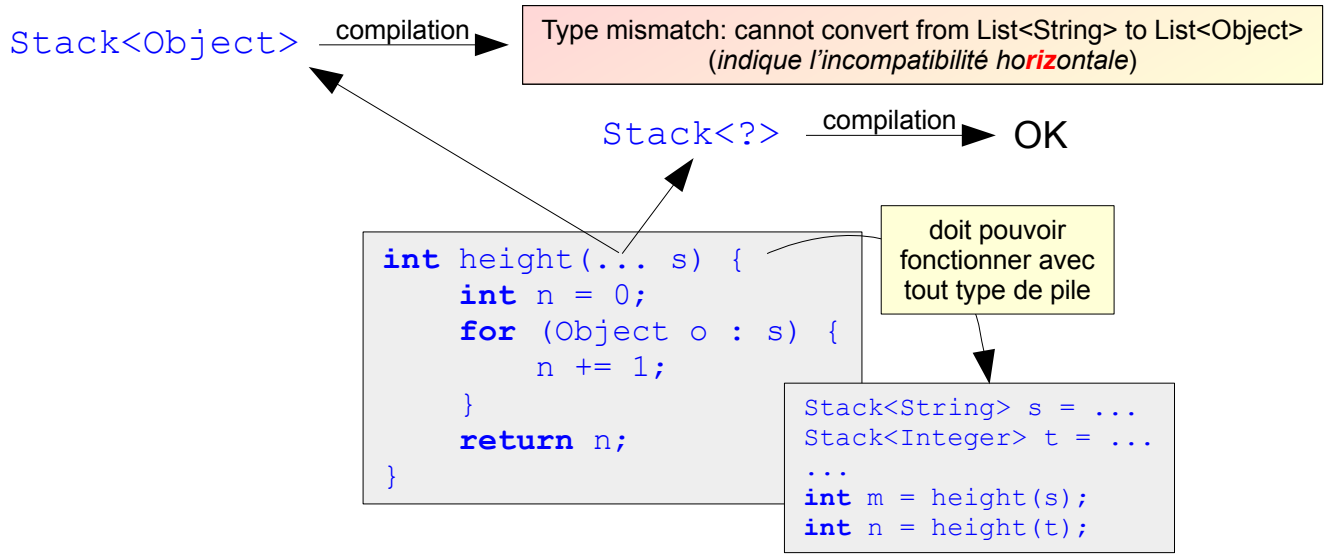
```
class RomanNumber extends Number implements CharSequence> {
    // des nombres entiers qui s'écrivent avec
    // des I des V des X des L des C des D et des M
    // par exemple MCMXXCVI ou MMXXIII
}
```

`G<RomanNumber>` est légal
`G<Integer>` n'est pas légal

Généricité
III. Généricité contrainte
1. Contrainte générique
2. Bornes multiples
IV. Joker
1. Définition
2. Bornes du joker
3. Types à joker borné
4. Sous-typage
5. Instanciation des types paramétrés
V. Méthodes génériques
1. Motivation
2. Définition
3. Inférence des paramètres de types
i. Algorithme d'inférence
ii. Exemple 1
iii. Exemple 2
iv. Exemple 3
4. Capture du joker
i. Motivation et définition
ii. Exemples
5. Règles méthodologiques
i. Méthode générique ou à joker ?
ii. Utilisation des jokers bornés
a. Règles 1 & 2
b. Règles 3 & 4
VI. Compléments
1. Tableaux et généricité
2. Exceptions et généricité
3. Classe Class<E>
4. Types énumérés
i. Classe Enum<E>
ii. Entête de la classe Enum
iii. Exemple élaboré

Joker de type : symbole utilisé dans une instanciation générique pour représenter le super-type commun aux types paramétrés d'un même type générique.





Généricité

III. Généricité contrainte
1. Contrainte générique
2. Bornes multiples
IV. Joker
1. Définition
2. Bornes du joker
3. Types à joker borné
4. Sous-typage
5. Instanciation des types paramétrés
V. Méthodes génériques
1. Motivation
2. Définition
3. Inférence des paramètres de types
i. Algorithme d'inférence
ii. Exemple 1
iii. Exemple 2
iv. Exemple 3
4. Capture du joker
i. Motivation et définition
ii. Exemples
5. Règles méthodologiques
i. Méthode générique ou à joker ?
ii. Utilisation des jokers bornés
a. Règles 1 & 2
b. Règles 3 & 4
VI. Compléments
1. Tableaux et généricité
2. Exceptions et généricité
3. Classe Class<E>
4. Types énumérés
i. Classe Enum<E>
ii. Entête de la classe Enum
iii. Exemple élaboré


```
abstract class Shape {
    abstract void draw(Canvas c);
}

class Circle extends Shape {
    void draw(Canvas c) {
        // dessine un cercle sur c
    }
}

class Rectangle extends Shape {
    void draw(Canvas c) {
        // dessine un rectangle sur c
    }
}
```

```
void m(Stack<Shape> p1,
       Stack<Circle> p2,
       Stack<Rectangle> p3) {
    Canvas c = new Canvas();
    c.drawAll(p1);
    c.drawAll(p2);
    c.drawAll(p3);
}
```

```
class Canvas {
    void draw(Shape s) {
        s.draw(this);
    }
    void drawAll(Stack<...> shapes) {
        for (Shape s: shapes) {
            s.draw(this);
        }
    }
}
```

Stack<Shape>

Stack<Rectangle>

Stack<Circle>

~~Stack<String>~~

Stack<? extends Shape>

BSup(?) = Shape

BInf(?) = ⊥

Stack<? extends Shape> est le super-type de tous les types paramétrés Stack<X> pour lesquels X est *sous-type* de Shape

BSup(?) = Object

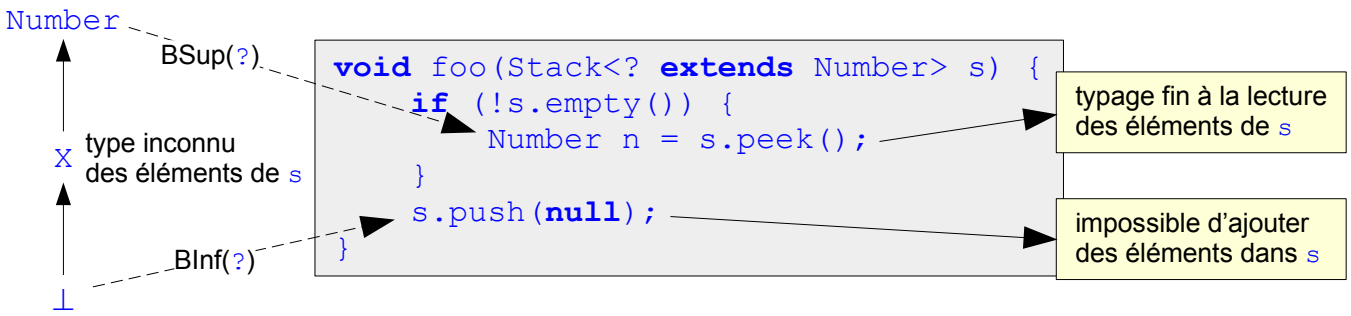
BInf(?) = Shape

Stack<? super Shape> est le super-type de tous les types paramétrés Stack<X> pour lesquels X est *super-type* de Shape

Généricité

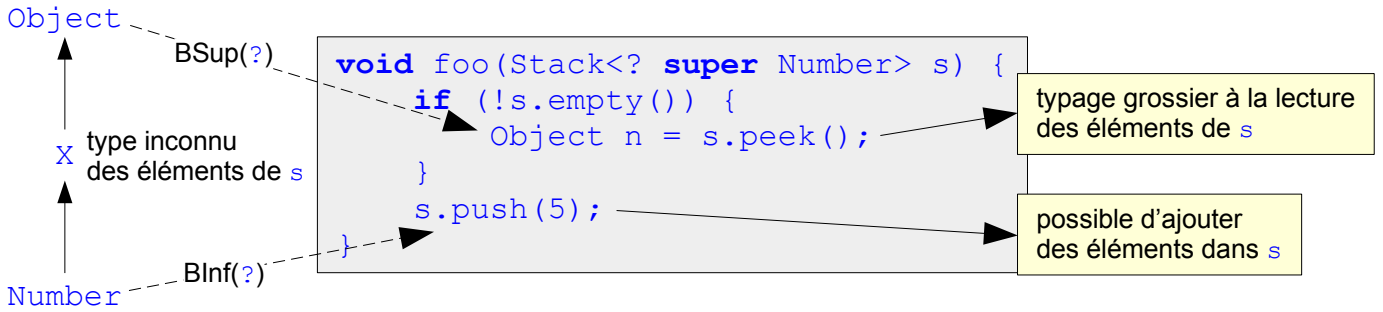
III. Généricité contrainte
1. Contrainte générique
2. Bornes multiples
IV. Joker
1. Définition
2. Bornes du joker
3. Types à joker borné
4. Sous-typage
5. Instanciation des types paramétrés
V. Méthodes génériques
1. Motivation
2. Définition
3. Inférence des paramètres de types
i. Algorithme d'inférence
ii. Exemple 1
iii. Exemple 2
iv. Exemple 3
4. Capture du joker
i. Motivation et définition
ii. Exemples
5. Règles méthodologiques
i. Méthode générique ou à joker ?
ii. Utilisation des jokers bornés
a. Règles 1 & 2
b. Règles 3 & 4
VI. Compléments
1. Tableaux et généricité
2. Exceptions et généricité
3. Classe Class<E>
4. Types énumérés
i. Classe Enum<E>
ii. Entête de la classe Enum
iii. Exemple élaboré

le type des éléments de `s` est compatible pour l'affectation avec les **super-types** de `Number`



les **sous-types** de ⊥ sont compatibles pour l'affectation avec `X` → (il n'y en a pas !)

le type des éléments de `s` est compatible pour l'affectation avec les **super-types** de `Object` → (il n'y en a pas !)

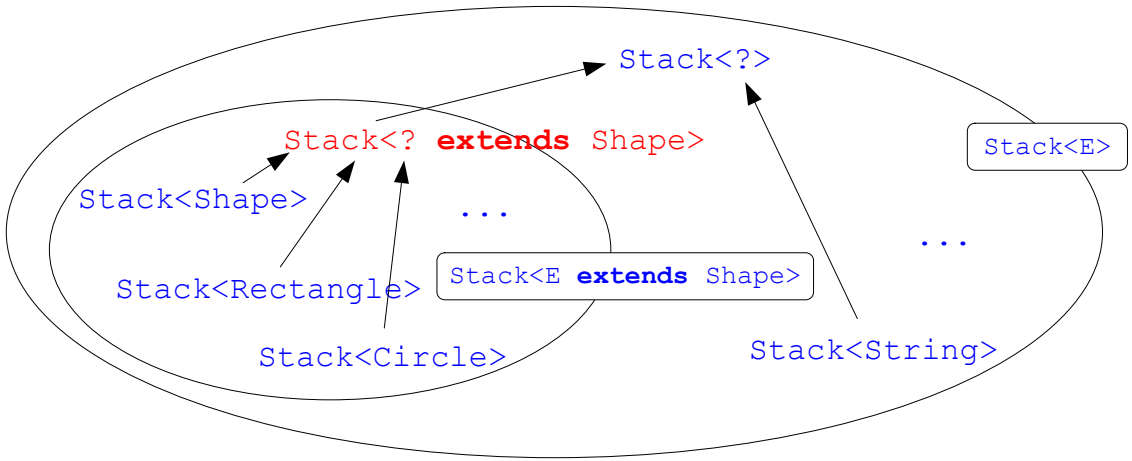


les **sous-types** de `Number` sont compatibles pour l'affectation avec `X`

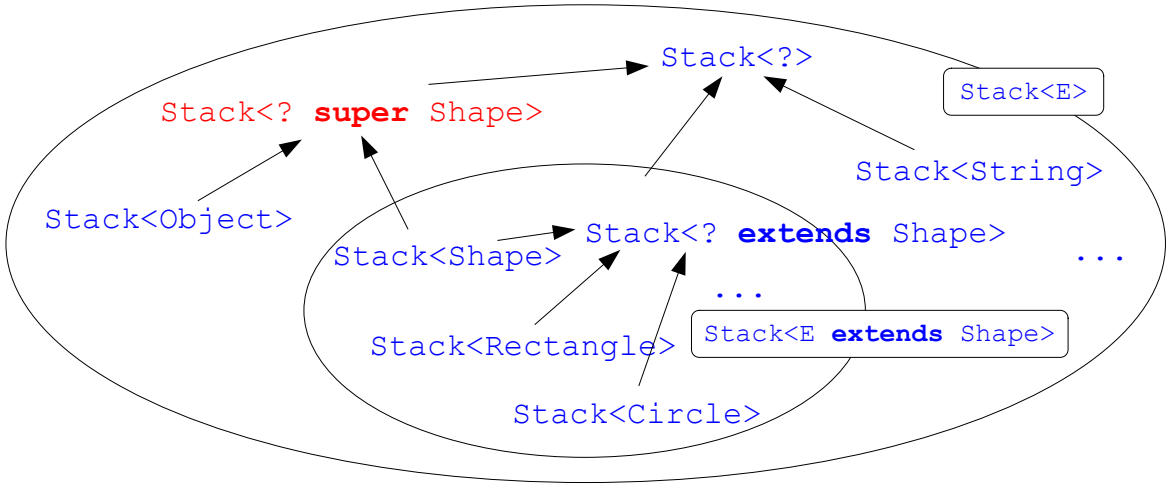
Généricité

III. Généricité contrainte
1. Contrainte générique
2. Bornes multiples
IV. Joker
1. Définition
2. Bornes du joker
3. Types à joker borné
4. Sous-typage
5. Instanciation des types paramétrés
V. Méthodes génériques
1. Motivation
2. Définition
3. Inférence des paramètres de types
i. Algorithme d'inférence
ii. Exemple 1
iii. Exemple 2
iv. Exemple 3
4. Capture du joker
i. Motivation et définition
ii. Exemples
5. Règles méthodologiques
i. Méthode générique ou à joker ?
ii. Utilisation des jokers bornés
a. Règles 1 & 2
b. Règles 3 & 4
VI. Compléments
1. Tableaux et généricité
2. Exceptions et généricité
3. Classe Class<E>
4. Types énumérés
i. Classe Enum<E>
ii. Entête de la classe Enum
iii. Exemple élaboré

$\forall$ X <: Shape, Stack<X> <: Stack<? **extends** Shape>

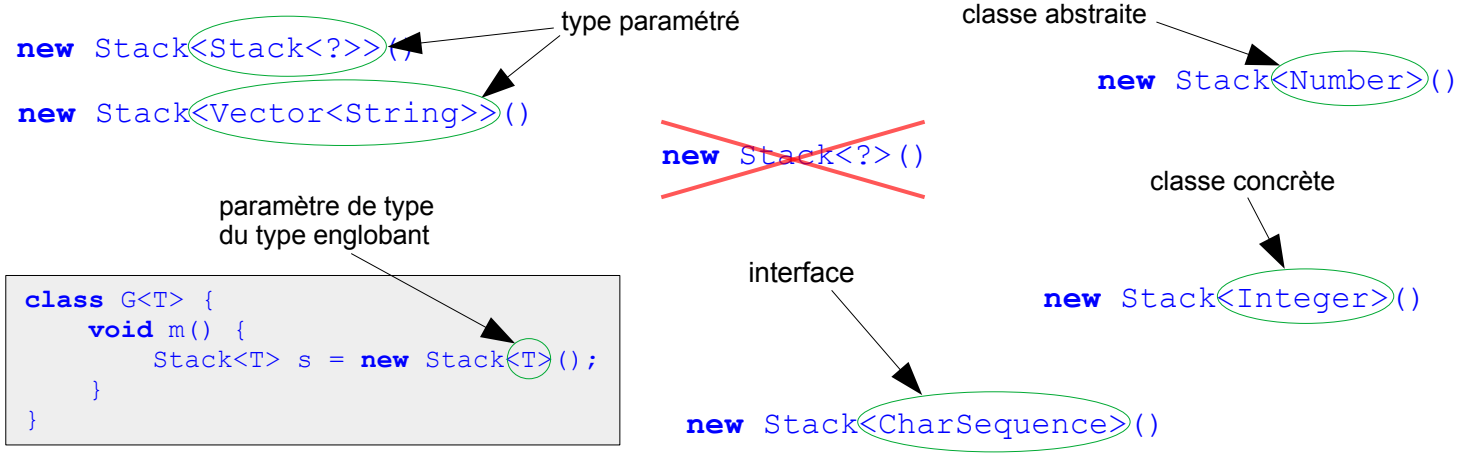


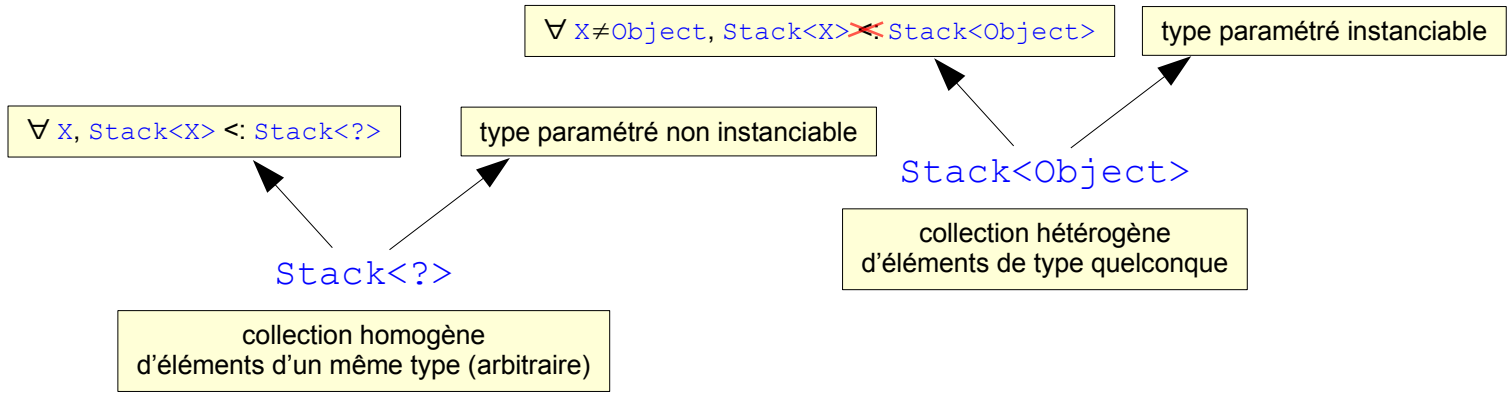
$\forall$ X :> Shape, Stack<X> <: Stack<? **super** Shape>



Généricité
III. Généricité contrainte
1. Contrainte générique
2. Bornes multiples
IV. Joker
1. Définition
2. Bornes du joker
3. Types à joker borné
4. Sous-typage
5. Instanciation des types paramétrés
V. Méthodes génériques
1. Motivation
2. Définition
3. Inférence des paramètres de types
i. Algorithme d'inférence
ii. Exemple 1
iii. Exemple 2
iv. Exemple 3
4. Capture du joker
i. Motivation et définition
ii. Exemples
5. Règles méthodologiques
i. Méthode générique ou à joker ?
ii. Utilisation des jokers bornés
a. Règles 1 & 2
b. Règles 3 & 4
VI. Compléments
1. Tableaux et généricité
2. Exceptions et généricité
3. Classe Class<E>
4. Types énumérés
i. Classe Enum<E>
ii. Entête de la classe Enum
iii. Exemple élaboré

Un *type paramétré* est *instanciable* ssi :
- son type de base est instanciable, et
- ses paramètres effectifs sont tous différents de ?





Généricité
III. Généricité contrainte
1. Contrainte générique
2. Bornes multiples
IV. Joker
1. Définition
2. Bornes du joker
3. Types à joker borné
4. Sous-typage
5. Instanciation des types paramétrés
V. Méthodes génériques
1. Motivation
2. Définition
3. Inférence des paramètres de types
i. Algorithme d'inférence
ii. Exemple 1
iii. Exemple 2
iv. Exemple 3
4. Capture du joker
i. Motivation et définition
ii. Exemples
5. Règles méthodologiques
i. Méthode générique ou à joker ?
ii. Utilisation des jokers bornés
a. Règles 1 & 2
b. Règles 3 & 4
VI. Compléments
1. Tableaux et généricité
2. Exceptions et généricité
3. Classe Class<E>
4. Types énumérés
i. Classe Enum<E>
ii. Entête de la classe Enum
iii. Exemple élaboré

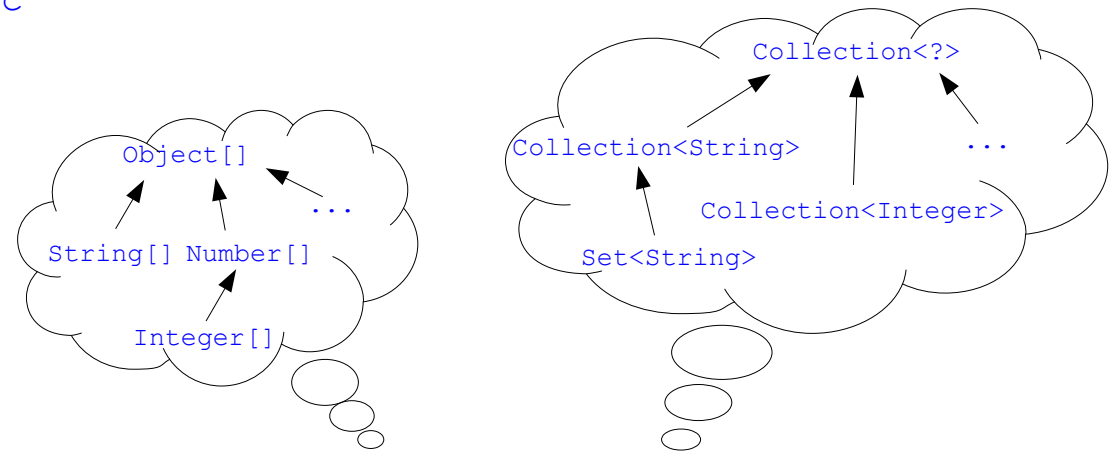
Comment coder une méthode
qui ajoute tous les éléments d'un tableau dans une collection
avec une signature utilisable dans le plus de cas possibles ?

```
String[] sa = new String[] { "un", "deux" };
Integer[] ia = new Integer[] { 1, 2 };
Collection<String> sc = new ArrayList<String>();
Collection<Integer> ic = new ArrayList<Integer>();
Set<String> ss = new HashSet<String>();

a.addAll(sa, sc);
a.addAll(ia, ic);
a.addAll(sa, ss);
```

t c

```
class A {
    void addAll(...[] t, Collection<...> c) {
        for (... obj : t) {
            c.add(obj);
        }
    }
}
```




```
for (Object obj : t) {
    c.add(obj);
}
```

```
a.addAll(sa, ic);
```

Généricité
III. Généricité contrainte
1. Contrainte générique
2. Bornes multiples
IV. Joker
1. Définition
2. Bornes du joker
3. Types à joker borné
4. Sous-typage
5. Instanciation des types paramétrés
V. Méthodes génériques
1. Motivation
2. Définition
3. Inférence des paramètres de types
i. Algorithme d'inférence
ii. Exemple 1
iii. Exemple 2
iv. Exemple 3
4. Capture du joker
i. Motivation et définition
ii. Exemples
5. Règles méthodologiques
i. Méthode générique ou à joker ?
ii. Utilisation des jokers bornés
a. Règles 1 & 2
b. Règles 3 & 4
VI. Compléments
1. Tableaux et généricité
2. Exceptions et généricité
3. Classe Class<E>
4. Types énumérés
i. Classe Enum<E>
ii. Entête de la classe Enum
iii. Exemple élaboré

Méthode générique : famille de méthodes, paramétrée par une ou plusieurs variables (ou paramètres) de types.

[*Access*] [*Modif*] <DT_1,...,DT_n> *Type Signature* [*Corps*]

DT_i → T_i **extends** $U_{i,1}$ & ... & U_{i,k_i} ($1 \le i \le n$ et $k_i \ge 0$)

```
class A<T> {
    void m1(T x) { ... }
    void m2(Stack<String> x) { ... }
    void m3(Stack<?> x) { ... }
    <S> void m4(S x) { ... }
    static <S> void m5(S x) { ... }
}
```

```
class B {
    void m1() { ... }
    void m2(Stack<String> x) { ... }
    void m3(Stack<?> x) { ... }
    <S> void m4(S x) { ... }
    static <S> void m5(S x) { ... }
}
```

méthodes génériques

```
<T> void addAll(T[] tab, Collection<T> c) {
    for (T x : tab) {
        c.add(x);
    }
}
```

utilisations du paramètre de type

déclaration du paramètre de type de la méthode générique

instanciations génériques

T ::= Integer

```
Integer[] ia = ...;
ArrayList<Integer> ial = ...;
obj.<Integer>addAll(ia, ial);
```

T ::= String

```
String[] sa = ...;
HashSet<String> shs = ...;
obj.<String>addAll(sa, shs);
```