

Rappels et compléments

V. Héritage

1. Mécanisme d'héritage
 - i. Définition générale
 - ii. Relation entre classes
 - iii. Formes d'héritage en Java
2. Sous-type Java
 - i. Sous-type Java direct
 - ii. Sous-type Java
 - iii. Sous-types tableaux
3. Expressions
 - i. Définitions
 - ii. Valeur d'une expression
 - iii. Types d'une expression
 - iv. Transtypage
 - a. Définition
 - b. Extensibilité potentielle
4. Masquage d'attribut
5. Chaînage des constructeurs
6. Redéfinition de méthode
 - i. Redéf° et liaison dynamique
 - ii. Modification d'entête
 - iii. @Override
 - iv. Redéfinition et surcharge
7. Invocation de méthode
 - i. Principe
 - a. Méthode virtuelle
 - b. Méthode de classe
 - c. Méthode privée
 - d. Méthode avec super
 - ii. Invocation et surcharge
 - a. Résolution d'appel
 - b. Surcharge et héritage
 - c. Piège de la surcharge
 - d. Ambiguïté
8. Accessibilité
 - i. Paquetage
 - ii. Accessibilité des types
 - iii. Accessibilité des caract.

Type d'une expression (*Compile-time Type*) : type calculé selon les règles

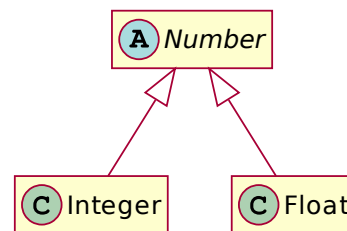
- littéral → type défini dans le langage
- opérateur → type de la valeur produite défini dans le langage
- variable → type déclaré dans le code
- appel de méthode → type de retour déclaré dans le code

$CT(3) = \text{int}$

$CT(3.0) = \text{double}$

$CT(c_1 \ \&\& \ c_2) = \text{boolean}$

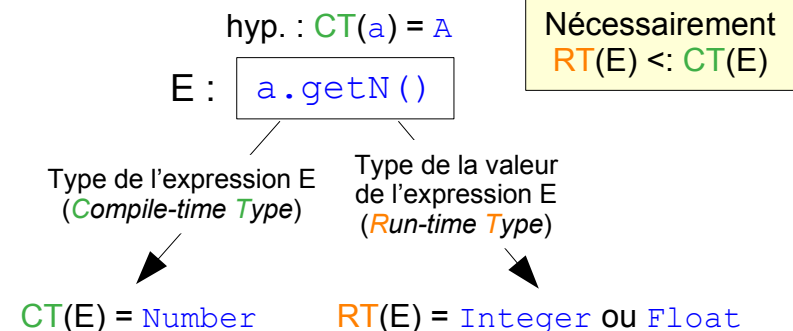
$\text{int } v$
 $CT(v) = \text{int}$



```
class A {
    private Number n;
    A(Integer i) { n = i; }
    A(Float f) { n = f; }
    public Number getN() {
        return n;
    }
}
```

Type de la valeur d'une expression (*Run-time Type*) :

- valeur primitive → CT de l'expression
- valeur référence → classe génératrice de l'objet



Opérateur instanceof : détermine, pendant l'exécution, si la valeur d'une expression est une instance du type indiqué.

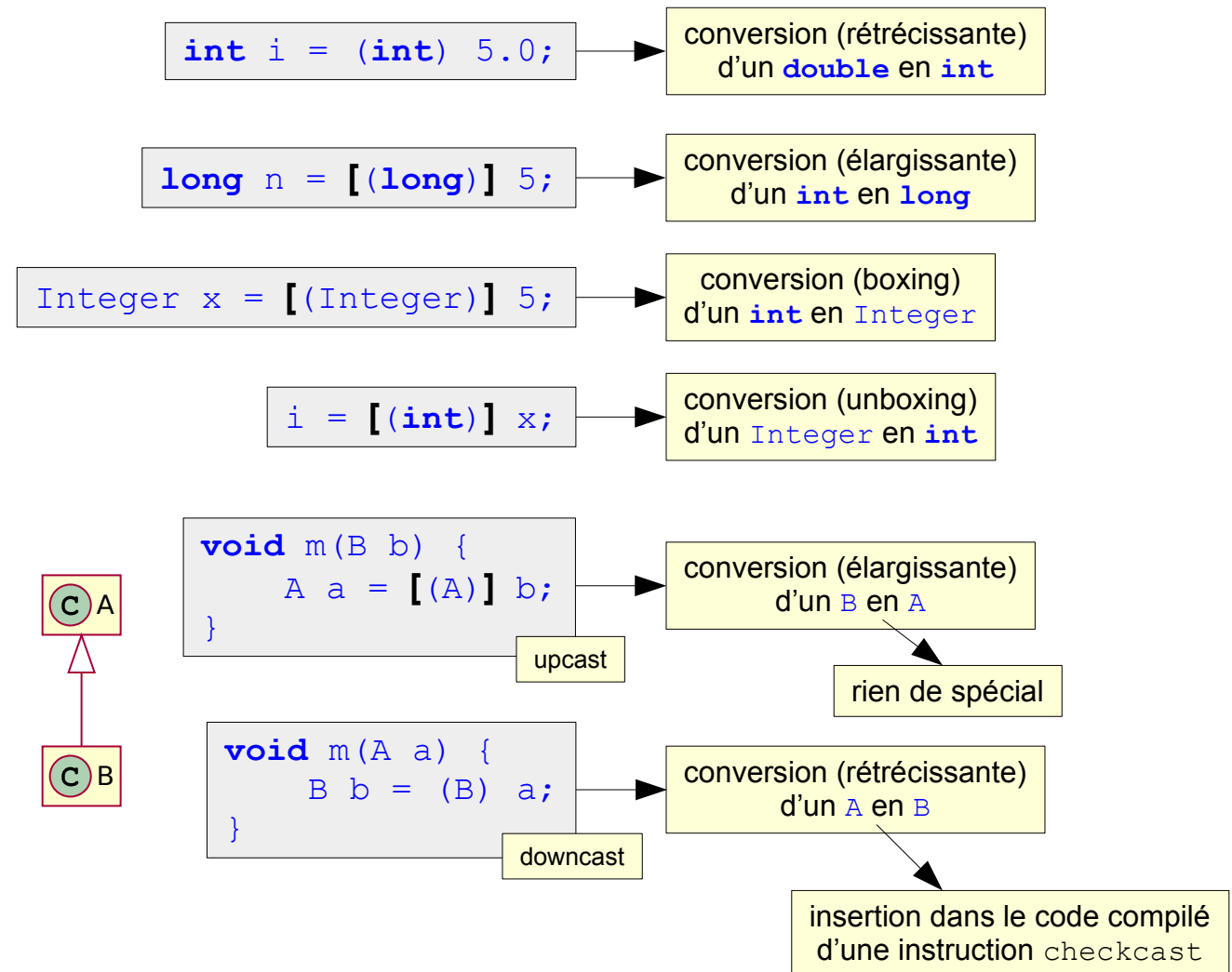
```
if (e instanceof Integer) {
    // true : e != null et e est une instance de Integer (RT(e) <: Integer)
} else {
    // false : e == null ou e n'est pas une instance de Integer
}
```

Rappels et compléments

V. Héritage

1. Mécanisme d'héritage
 - i. Définition générale
 - ii. Relation entre classes
 - iii. Formes d'héritage en Java
2. Sous-typage Java
 - i. Sous-type Java direct
 - ii. Sous-type Java
 - iii. Sous-types tableaux
3. Expressions
 - i. Définitions
 - ii. Valeur d'une expression
 - iii. Types d'une expression
 - iv. Transtypage
 - a. Définition
 - b. Extensibilité potentielle
4. Masquage d'attribut
5. Chaînage des constructeurs
6. Redéfinition de méthode
 - i. Redéf° et liaison dynamique
 - ii. Modification d'entête
 - iii. @Override
 - iv. Redéfinition et surcharge
7. Invocation de méthode
 - i. Principe
 - a. Méthode virtuelle
 - b. Méthode de classe
 - c. Méthode privée
 - d. Méthode avec super
 - ii. Invocation et surcharge
 - a. Résolution d'appel
 - b. Surcharge et héritage
 - c. Piège de la surcharge
 - d. Ambiguïté
8. Accessibilité
 - i. Paquetage
 - ii. Accessibilité des types
 - iii. Accessibilité des caract.

Transtypage (cast) : remplacement du type d'une expression par un autre type, à l'aide d'une opération de cast « *(IdType) e* ».



Rappels et compléments

mémorisez cette définition pour suivre la prochaine diapo !

V. Héritage

1. Mécanisme d'héritage
 - i. Définition générale
 - ii. Relation entre classes
 - iii. Formes d'héritage en Java
2. Sous-typage Java
 - i. Sous-type Java direct
 - ii. Sous-type Java
 - iii. Sous-types tableaux
3. Expressions
 - i. Définitions
 - ii. Valeur d'une expression
 - iii. Types d'une expression
- iv. Transtypage
 - a. Définition
 - b. Extensibilité potentielle
4. Masquage d'attribut
5. Chaînage des constructeurs
6. Redéfinition de méthode
 - i. Redéf° et liaison dynamique
 - ii. Modification d'entête
 - iii. @Override
 - iv. Redéfinition et surcharge
7. Invocation de méthode
 - i. Principe
 - a. Méthode virtuelle
 - b. Méthode de classe
 - c. Méthode privée
 - d. Méthode avec super
 - ii. Invocation et surcharge
 - a. Résolution d'appel
 - b. Surcharge et héritage
 - c. Piège de la surcharge
 - d. Ambiguïté
8. Accessibilité
 - i. Paquetage
 - ii. Accessibilité des types
 - iii. Accessibilité des caract.

Extensibilité potentielle :

S est potentiellement extensible en un type compatible avec T si

- S n'est pas un sous-type Java de T, mais
- il **pourrait** exister à l'exécution un sous-type commun à S et T.

```
void m(S y) {
    T x = (T) y;
}
```

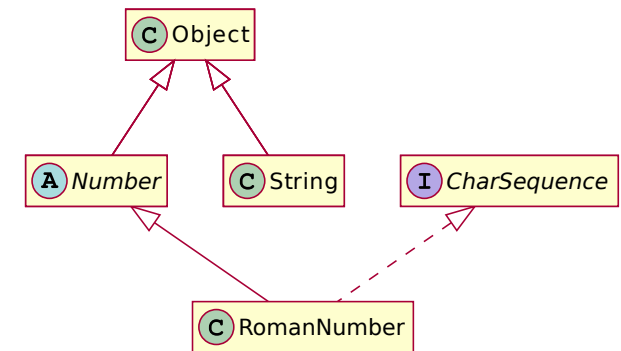
~~S <: T~~ mais serait-il possible que
RT(y) <: S et RT(y) <: T ?

```
void m(String y) {
    Number x = (Number) y;
}
```

String **n'est pas** potentiellement extensible en Number

```
void m(CharSequence y) {
    Number x = (Number) y;
}
```

CharSequence **est** potentiellement extensible en Number



```
void m(S y) {
    T x = (T) y;
}
```

compilation

S <: T

statiquement **correct**
pas de test
dans le bytecode

upcast → automatique

aload_1
astore_2

ext. pot.

statiquement **invérifiable**
insertion de checkcast
dans le bytecode

si S >: T c'est
un **downcast**

aload_1
checkcast
astore_2

~~S <: T~~
~~ext. pot.~~

statiquement **incorrect**
erreur : cast impossible

Rappels et compléments

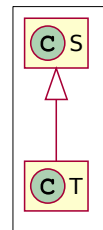
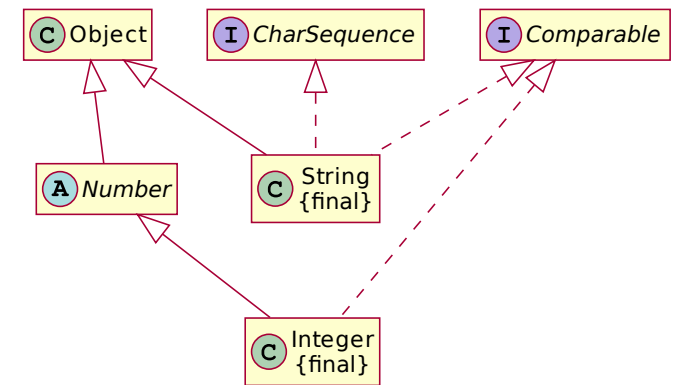
V. Héritage

1. Mécanisme d'héritage
 - i. Définition générale
 - ii. Relation entre classes
 - iii. Formes d'héritage en Java
2. Sous-typage Java
 - i. Sous-type Java direct
 - ii. Sous-type Java
 - iii. Sous-types tableaux
3. Expressions
 - i. Définitions
 - ii. Valeur d'une expression
 - iii. Types d'une expression
 - iv. Transtypage
 - a. Définition
 - b. Extensibilité potentielle
4. Masquage d'attribut
5. Chaînage des constructeurs
6. Redéfinition de méthode
 - i. Redéf° et liaison dynamique
 - ii. Modification d'entête
 - iii. @Override
 - iv. Redéfinition et surcharge
7. Invocation de méthode
 - i. Principe
 - a. Méthode virtuelle
 - b. Méthode de classe
 - c. Méthode privée
 - d. Méthode avec super
 - ii. Invocation et surcharge
 - a. Résolution d'appel
 - b. Surcharge et héritage
 - c. Piège de la surcharge
 - d. Ambiguïté
8. Accessibilité
 - i. Paquetage
 - ii. Accessibilité des types
 - iii. Accessibilité des caract.

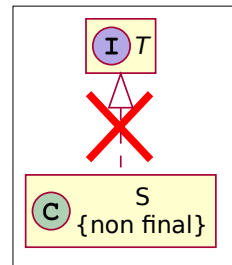
```
void m(S y) {
    T x = (T) y;
}
```

sera-t-il possible que
RT(y) <: S et RT(y) <: T ?

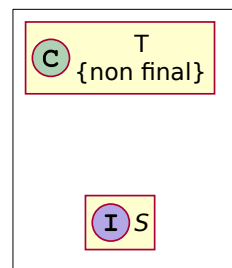
Il n'y a que 6 cas d'extensibilité potentielle



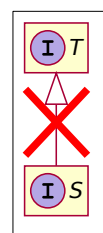
```
void m(Object y) {
    Number x = (Number) y;
}
RT(y) = Integer
```



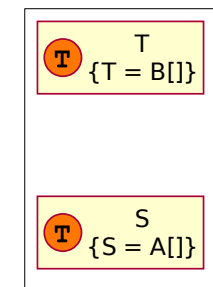
```
void m(Number y) {
    Comparable x = (Comparable) y;
}
RT(y) = Integer
```



```
void m(Comparable y) {
    Number x = (Number) y;
}
RT(y) = Integer
```

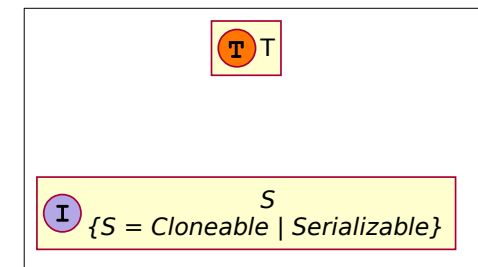


```
void m(Comparable y) {
    CharSequence x = (CharSequence) y;
}
RT(y) = String
```



A pot. ext.
en un type
compatible
avec B

```
void m(Comparable[] y) {
    Number[] x = (Number[]) y;
}
RT(y) = Integer[]
```



```
void m(Cloneable y) {
    Object[] x = (Object[]) y;
}
RT(y) = Integer[]
```

Rappels et compléments

V. Héritage

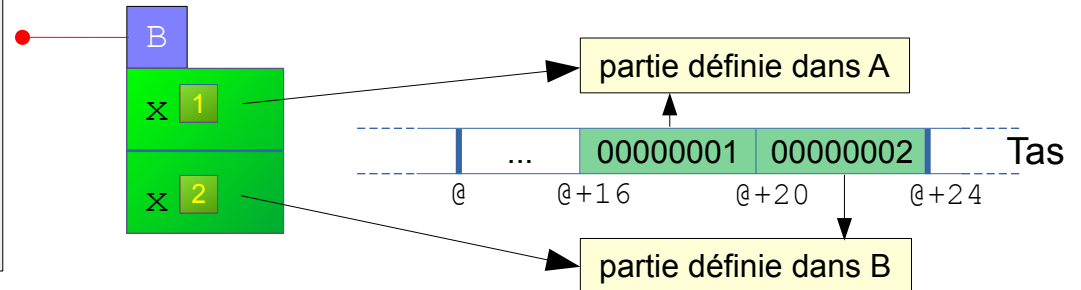
- Mécanisme d'héritage
 - Définition générale
 - Relation entre classes
 - Formes d'héritage en Java
- Sous-typage Java
 - Sous-type Java direct
 - Sous-type Java
 - Sous-types tableaux
- Expressions
 - Définitions
 - Valeur d'une expression
 - Types d'une expression
- Transtypage
 - Définition
 - Extensibilité potentielle
- Masquage d'attribut
- Chaînage des constructeurs
- Redéfinition de méthode
 - Redéf° et liaison dynamique
 - Modification d'entête
 - @Override
 - Redéfinition et surcharge
- Invocation de méthode
 - Principe
 - Méthode virtuelle
 - Méthode de classe
 - Méthode privée
 - Méthode avec super
 - Invocation et surcharge
 - Résolution d'appel
 - Surcharge et héritage
 - Piège de la surcharge
 - Ambiguïté
- Accessibilité
 - Paquetage
 - Accessibilité des types
 - Accessibilité des caract.

Masquage d'attribut

La redéclaration d'un attribut dans une sous-classe masque l'attribut correspondant dans la super-classe.

Résolution d'accès : c'est le type de l'expression cible qui détermine l'attribut auquel on accède.

```
class A {  
    int x = 1;  
}  
  
class B extends A {  
    int x = 2;  
}
```



```
B u = new B();  
System.out.println(u.x); // 2  
A v = u;  
System.out.println(v.x); // 1
```

CT(u) = B ⇒ u.x sélectionne le x de B

CT(v) = A ⇒ v.x sélectionne le x de A

```
System.out.println((A) u).x); // 1
```



CT((A) u) = A

```
class A {  
    int x = 1;  
    void m1() {  
        System.out.println(this.x);  
    }  
}  
  
class B extends A {  
    int x = 2;  
    void m2() {  
        System.out.println(this.x);  
        System.out.println(super.x);  
    }  
}
```



dans le code de A,
CT(this) = A

```
B u = new B();  
u.m1(); // 1  
u.m2(); // 2 1
```



dans le code de B,
CT(this) = B
CT(super) = A