BIENVENUE

TECHNIFUTUR®
CENTRE DE COMPETENCES

www.enmieux.be

FEDER-FSE
UNION EUROPEENNE

Wallonie

L'UNION EUROPÉENNE ET LA WALLONIE
INVESTISSENT DANS VOTRE AVENIR

JAVA BASE

Les type de données primitifs

TYPES PRIMITIFS

- Booléen
  - boolean : true, false
- Entier signé
  - byte : de -128 à 127
    - (codé sur 8 bits ou 1 octets)
  - short : de -32.768 à 32.767
    - (codé sur 16 bits ou 2 octets)
  - int : de -2 147 483 648 à 2 147 483 647
    - (codé sur 32 bits ou 4 octets)
  - long : de -9.223.372.036.854.775 808 à 9.223.372.036.854.775.807
    - (codé sur 64 bits ou 8 octets)
- Flottant
  - float : de -1.40239846 E -45 à 3.40282347 E 38
    - (codé sur 32 bits ou 4 octets)
  - double : de 4.9406564584124654 E -324 à 1.797693134862316 E 308
    - (codé sur 64 bits ou 8 octets)
- Charactère
  - char : de \u0000 à \uFFFF (Unicode Plan multilingue de base)
    - (codé sur 16 bits ou 2 octets)

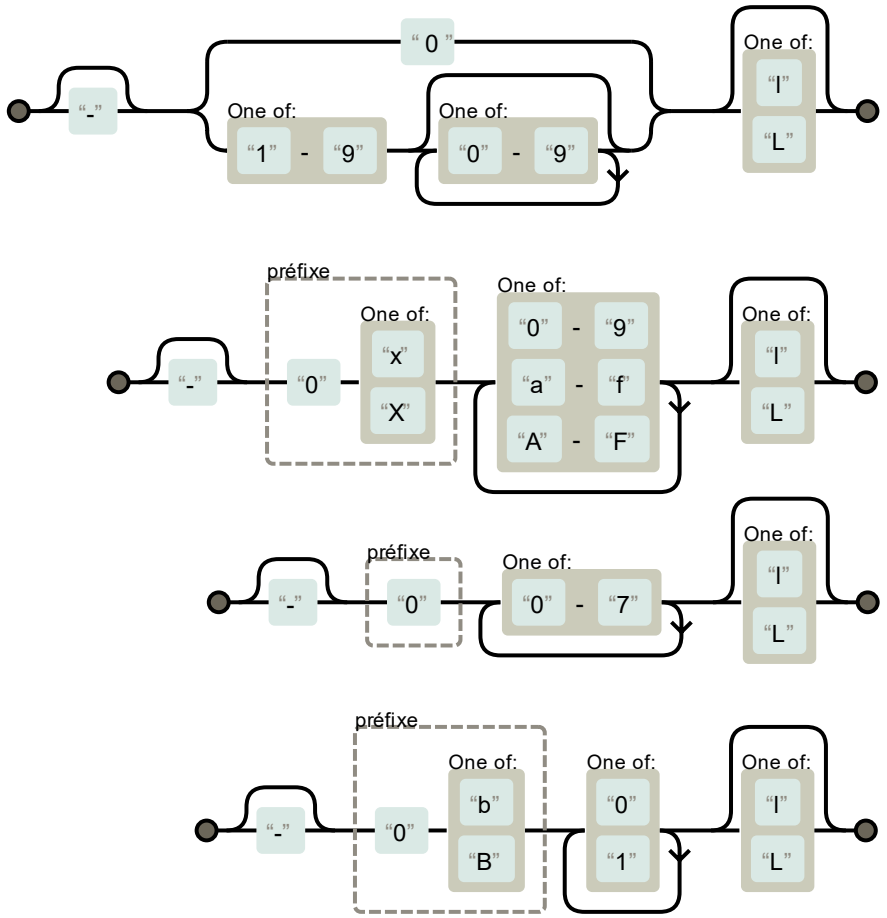
BOOLEAN

- Littéraux :
  - true
  - false
- Casting :
  - Pas de casting vers ou à partir d'un boolean.
- Autoboxing et unboxing vers Boolean
- Opérations
  - Relationnelles : == (egal), != (différent)
  - Logiques : ! (non), & (et), | (ou), ^ (ou exclusif)
  - Logiques à évaluation courte : && (et), || (ou)
  - Conditionnel : ? :
  - Concaténation : + (lors d'un concaténation avec une String)
    - Transformation du booléen en String
      - new Boolean(x).toString())

TECHNIFUTUR
CENTRE DE COMPETENCES

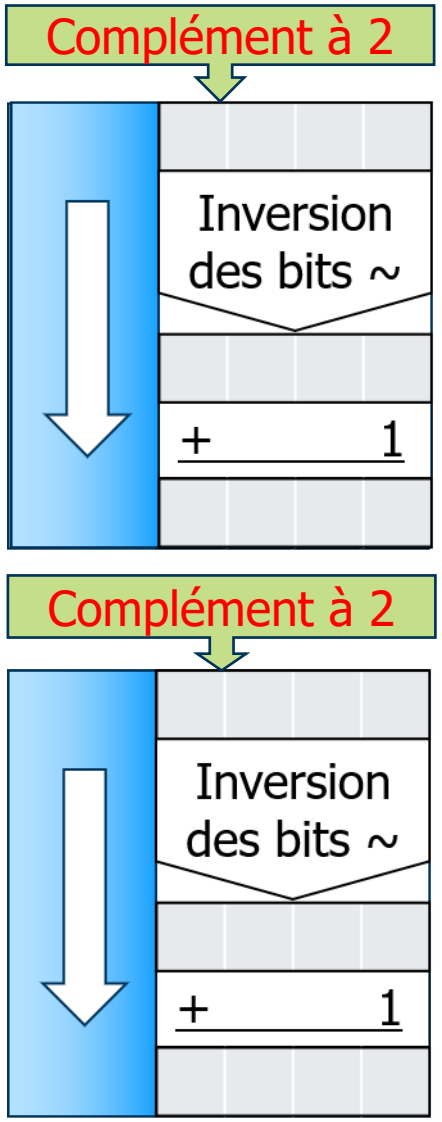
Les littéraux

- (avec le suffixe 'l'ou 'L' long sinon int)
  - Écriture décimale:
    - (ex: -158, 0, 32)
    - (regex: -?(0|[1-9][0-9]*)[lL]?)
  - Écriture hexadécimale :
    - (ex: 0xC01, -0xCAFE)
    - (regex: -?(0[xX])[0-9a-fA-F]+[lL]?)
  - Écriture octales :
    - (ex: 021, 01)
    - (regex: -?(0)[0-7]+[lL]?)
  - Écriture binaires :
    - (ex 0b11001, -0B001101)
    - (regex: -?(0[bB])[01]+[lL]?)
- Remarque :
  - Il possible d'insérer entre les chiffres des '_' pour faciliter la lecture du nombre



ENTIER SIGNÉ (COMPLÉMENT À 2)

0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				



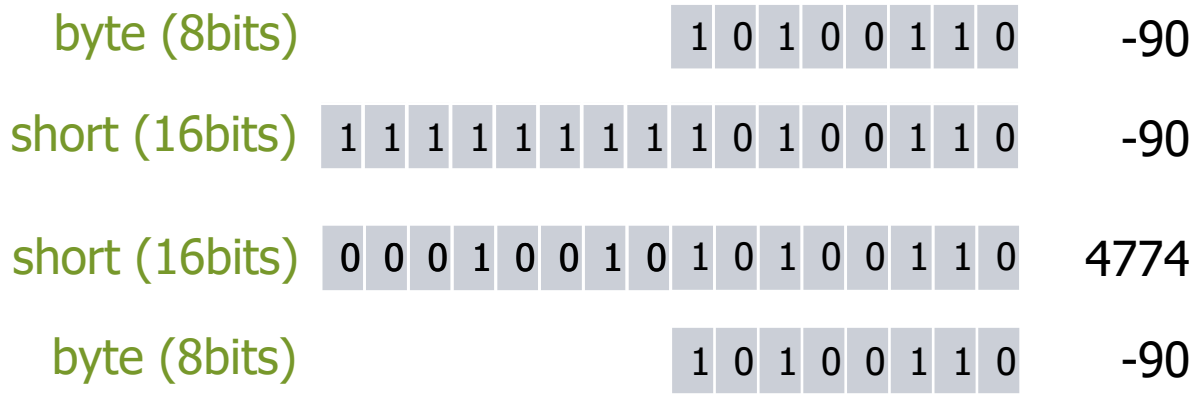
-8	1	0	0	0
-7	1	0	0	1
-6	1	0	1	0
-5	1	0	1	1
-4	1	1	0	0
-3	1	1	0	1
-2	1	1	1	0
-1	1	1	1	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1

ENTIERS

- Casting
  - Automatique :
    - byte -> short, int, long, float, double
    - short -> int, long, float, double
    - int -> long, float, double
    - long -> float, double
  - Casting automatique lors des opérations entre des entiers
    - Si un des opérandes est un long la valeur de l'autre opérande est transformé en long
    - Sinon les 2 opérandes sont transformés si nécessaire en int
    - Aucune opération n'est effectuée en dessous du int
  - Avec l'opérateur de casting (attention overflow)
    - Vers et à partir de tous les types primitifs sauf boolean
- AutoBoxing et unboxing
  - byte <-> Byte ,
  - short <-> Short,
  - int <-> Integer,
  - long <-> Long

CASTING DES ENTIERS ÉCRITURE BINAIRE

- Vers un type plus grand
  - Copie vers les bits de poids faible
  - Propagation du bit de signe
- Vers un type plus petit (casting explicite)
  - Copie vers les bits de poids faible
  - Perte des bits de poids fort



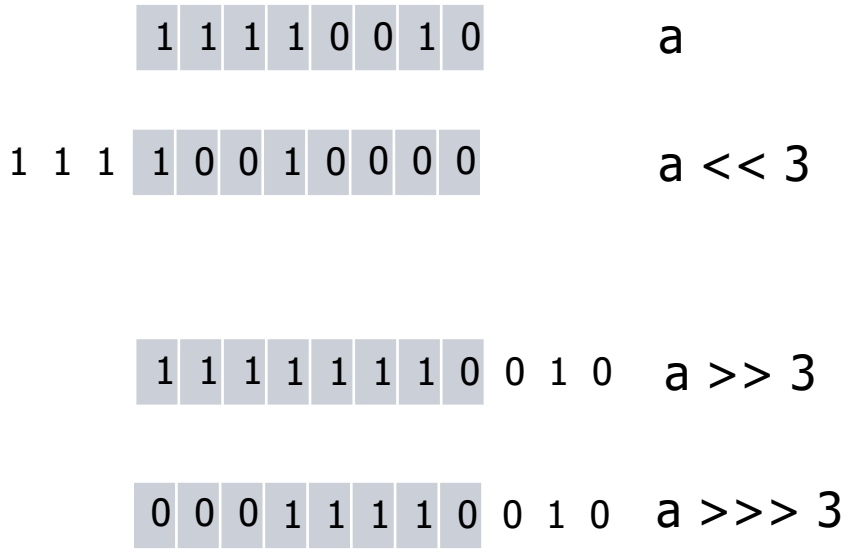
TECHNIFUTUR
CENTRE DE COMPETENCES

- Opérateurs
  - De comparaison <, <=, >, >=, ==, !=
  - Arithmètique
    - Les opérateurs unaire de signe + et –
    - De multiplication et de division *, /, et %
    - D'addition + et –
  - A effet de bord
    - Pré et post incrementation ++
    - Pré et post décrémentation --
  - Orienté écriture binaire
    - Décalage signé et non signé de bits : <<, >>, et >>>
    - Inversion des bits : ~
    - Travaillant bit à bit : &, ^, et |
  - Concaténation : + (lors d'un concaténation avec une String)
    - Transformation de l'entier en String
      - String.valueOf(int i)
      - String.valueOf(long l)

ENTIER OPÉRATIONS ORIENTÉES ÉCRITURE BINAIRE

TECHNIFUTUR
CENTRE DE COMPETENCES

- Opérations de décalage binaire
  - byte a = 0b1111_0010;
  - Décalage vers les bits de poids forts
    - b = a << 3;
  - Décalage vers les bits de poids faible
    - avec propagation du bit de signe
      - b = a >> 3;
    - sans propagation du bit de signe
      - b = a >>> 3;
- Remarque
  - Ce code ne compile pas !
  - Pourquoi ?



EXEMPLE D'UTILISATION DU BINAIRE

java.lang.reflect.Modifier

Constant Field	Value
PUBLIC	1
PRIVATE	2
PROTECTED	4
STATIC	8
FINAL	16
SYNCHRONIZED	32
VOLATILE	64
TRANSIENT	128
NATIVE	256
INTERFACE	512
ABSTRACT	1024
STRICT	2048

java.util.Spliterator<T>

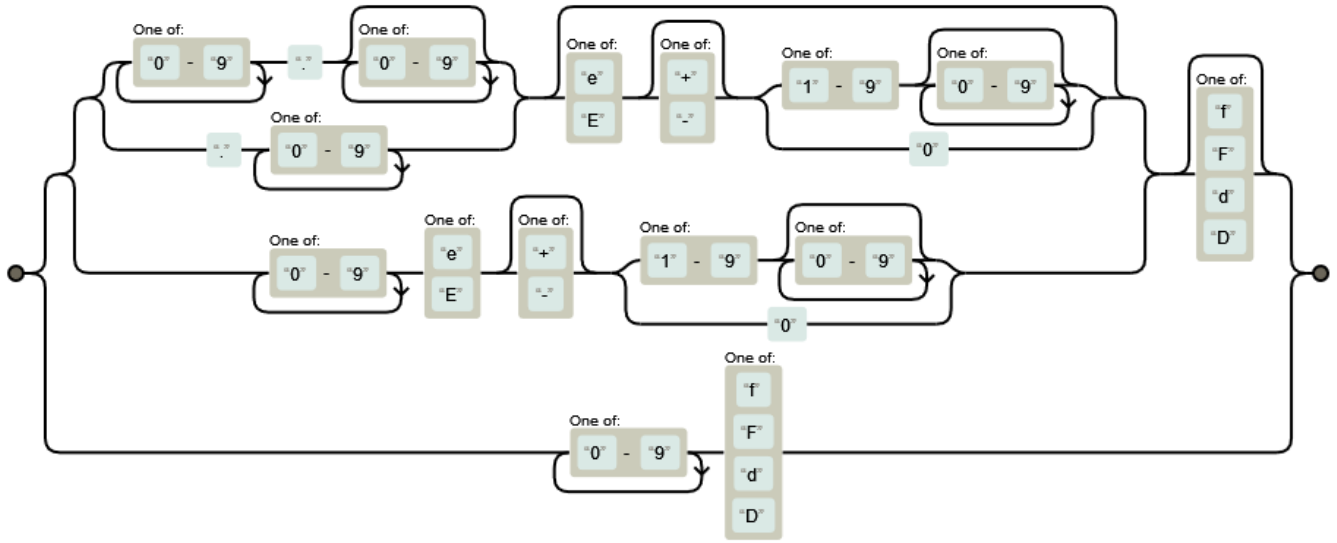
Constant Field	Value
DISTINCT	1
SORTED	4
ORDERED	16
SIZED	64
NONNULL	256
IMMUTABLE	1024
CONCURRENT	4096
SUBSIZED	16384

java.awt.event.InputEvent

Constant Field	Value
SHIFT_MASK	1
CTRL_MASK	2
BUTTON3_MASK	4
META_MASK	4
ALT_MASK	8
BUTTON2_MASK	8
BUTTON1_MASK	16
ALT_GRAPH_MASK	32
SHIFT_DOWN_MASK	64
CTRL_DOWN_MASK	128
META_DOWN_MASK	256
ALT_DOWN_MASK	512
BUTTON1_DOWN_MASK	1024
BUTTON2_DOWN_MASK	2048
BUTTON3_DOWN_MASK	4096
ALT_GRAPH_DOWN_MASK	8192

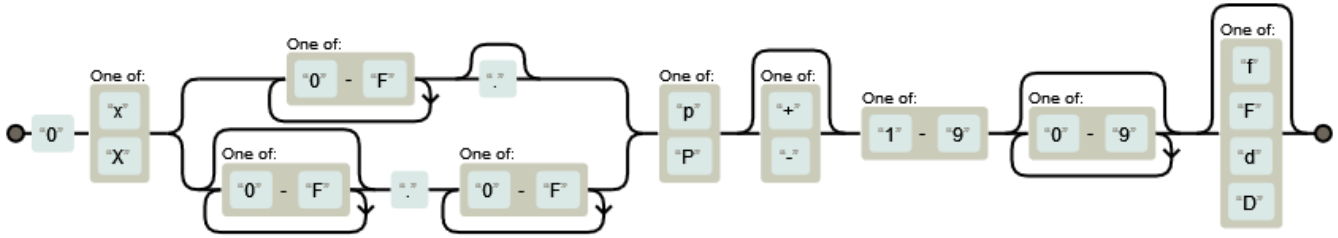
FLOTTANTS

- Les littéraux
  - Écriture décimale
    - Exemples :
      - 1.25
      - .32
      - -0.45e15
      - 10d
    - Regex :




```
(([0-9]+\.[0-9]*|\.[0-9]+)([eE][+-]?(?:[1-9][0-9]*|0))?|[0-9]+[eE][+-]?([1-9][0-9]*|0))[fFdD]?|[0-9]+[fFdD]
```

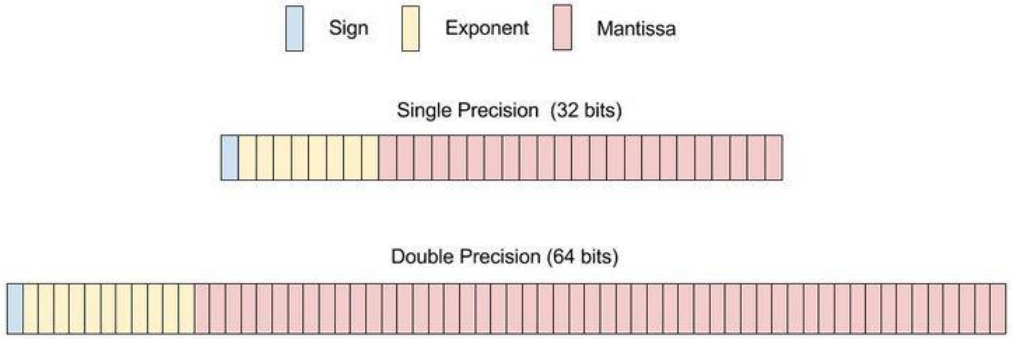
  - Écriture hexadécimale
    - Exemples :
      - 0xCAF.Ep-12f
    - Regex :




```
0[xX](?:[0-F]+\.?|[0-F]*\.[0-F]+)[pP][+-]?[1-9][0-9]*[fFdD]??
```

FLOTTANT

- Format binaire
  - float : coder sur 16 bits
  - double : coder sur 32 bits
  - Format IEEE 754
    - Notation scientifique
      - Approximation d'une valeur avec un certain niveau de précision
      - Certaine valeur ne peuvent pas être exprimé en binaire.
        - Ex 0.2 (1/5)
    - => attention applications comptables



- Casting
  - Automatique :
    - float -> double
  - Casting automatique lors des opérations
    - Si un des opérandes est un double la valeur de l'autre opérande est transformé en double
    - Sinon si un des opérandes est un float la valeur de l'autre opérande est transformé en float
    - Sinon voir règle sur les entiers
- AutoBoxing et unboxing
  - float <-> Float ,
  - double <-> Double

FLOTTANTS

TECHNIFUTUR
CENTRE DE COMPETENCES

Opérateurs

- De comparaison <, <=, >, >=, ==, !=
- Arithmètique
  - Les opérateurs unaire de signe + et –
  - De multiplication et de division *, /, et %
  - D'addition + et –
- A effet de bord
  - Pré et post incrementation ++
  - Pré et post décrémentation --
- Concaténation : + (lors d'un concaténation avec une String)
  - Transformation du flottant en String
    - String.valueOf(float f)
    - String.valueOf(double d)

CARACTÈRES

- Litéraux
  - unCaractère : ‘a’, ‘€’
  - séquenceEchapement : ‘\t’, ‘\n’
  - codeUTF-16 : ‘\u03a9’
    - Attention les codes unicode sont traduits avant l’interprétation du code
      - => ‘\u000a’ (LF) , ‘\u000d’ (CR), ‘\u0027’ (‘) impossible
- Format binaire
  - Entier non signé sur 16 bits
- Casting
  - Automatique :
    - char -> int, long, float, double
  - Casting automatique lors des opérations mathématiques
    - En int minimum
- AutoBoxing et unboxing
  - char <-> Character

PRIORITÉ DES OPÉRATEURS

Symbole	Note	Priorité	Associativité
() [] .	Parenthèse, accès tableau, accès méthode	17	Gauche à droite
++a --a	Préincrémentation, prédécrémentation	16	Droite à gauche
a++ a--	Postincrémentation, postdécrémentation	15	Gauche à droite
~	Inversion des bits d'un entier	14	Droite à gauche
!	Non logique pour un booléen	14	Droite à gauche
- +	Moins et plus unaire	14	Droite à gauche
(type)	Conversion de type (cast)	13	Droite à gauche
* / %	Opérations multiplicatives	12	Gauche à droite
- +	Opérations additives	11	Gauche à droite
<< >> >>>	Décalage de bits, à gauche et à droite	10	Gauche à droite
instanceof < <= > >=	Opérateurs relationnels	9	Gauche à droite
== !=	Opérateurs d'égalité	8	Gauche à droite
&	Et logique bit à bit	7	Gauche à droite
^	Ou exclusif logique bit à bit	6	Gauche à droite
|	Ou inclusif logique bit à bit	5	Gauche à droite
&&	Et conditionnel	4	Gauche à droite
||	Ou conditionnel	3	Gauche à droite
? :	Opérateur conditionnel	2	Droite à gauche
= *= /= %= += -= <<= >>= >>>= &= ^= |=	Opérateurs d'affectation	1	Droite à gauche
->	Flèche expression lambda	0	Droite à gauche

TECHNIFUTUR®
CENTRE DE COMPETENCES

MERCI DE VOTRE ATTENTION

WWW.TECHNIFUTUR.BE

VOS CONTACTS