Classes Enveloppes

Les classes enveloppes (*Wrappers*) sont des classes qui encapsulent les données de types primitifs, afin de pouvoir les utiliser comme des objets ordinaires. Les classes enveloppes sont définies dans le package standard de java (java.lang) importé par défaut. Les classes enveloppes sont *immuables*, c'est-à-dire que la valeur d'un objet créé à partir d'une classe enveloppe (Integer, Float, Double, …) ne peut pas être changée, pour donner une nouvelle valeur, il faudra créer un nouvel objet.

Type primitif	**Classe enveloppe**	
boolean	Boolean	
char	Character	
byte	Byte	Sous-classes de la classe *Number*
short	Short	
int	Integer	
long	Long	
float	Float	
double	Double	

Toutes les sous-classes de *Number* possèdent les méthodes *byte byteValue(), short shortValue(), int intValue(), long longValue* et *double doubleValue()* qui réalisent les conversions des types primitifs.

Toutes les classes enveloppes ont une méthode *x X.parseX(String s)* qui essaie de convertir une chaîne de caractères en une donnée de type primitif, si c'est possible (lèvent une exception *NumberFormatException* sinon). Pour les types entiers on a aussi les méthodes *x X.parseX(String s, int base)* qui essaient de convertir une chaîne de caractères en une donnée de type primitif, en supposant que la donnée est notée dans la base *base*.

Exemple

```
System.out.println(Integer.parseInt("234",7));
System.out.println(Byte.parseByte("234", 5));
```

affiche :

```
123
69
```

Toutes les classes enveloppes de type entier ont une méthode *static X X.decode(String s)* qui essaie de convertir une chaîne de caractères en une donnée de type primitif entier, si c'est possible (lèvent une exception *NumberFormatException* sinon).

- si la chaine commence par un 0 le décodage se fait en octal
- si la chaine commence par 0x, ou par # le décodage se fait en héxadécimal

- sinon le décodage se fait en décimal.

Exemple

```
System.out.println(Integer.decode("0234"));
System.out.println(Integer.decode("0xabf"));
System.out.println(Integer.decode("#abf"));
System.out.println(Integer.decode("234"));
```

affiche :

```
156
2751
2751
234
```

1. Boolean

Il y a deux constantes de type *Boolean* : *Boolean.TRUE* et *Boolean.FALSE*.

2. Character

Voici quelques unes des nombreuses méthodes de la classe *Character*.

static boolean isLetter(**char**c)	Retourne **true** si le caractère *c* est une lettre et **false** sinon.
static boolean isDigit(**char** c)	Retourne **true** si le caractère *c* est un chiffre et **false** sinon.
static boolean isLetterOrDigit(char c)	Retourne **true** si le caractère *c* est un chiffre ou une lettre et **false** sinon.
static boolean isLowerCase(**char** c)	Retourne **true** si le caractère *c* est une lettre majuscule et **false** sinon.
static boolean isUpperCase(**char** c)	Retourne **true** si le caractère *c* est un espace et **false** sinon.
static boolean isSpaceChar(**char** c)	Retourne **true** si le caractère *c* est un espace et **false** sinon.
static boolean isWhiteSpace(**char** c)	Retourne **true** si le caractère *c* est un espace selon Java et **false** sinon. ('\n', '\r', '\t', '\f' ...)
static boolean isJavaIdentifierPart(**char** c)	Retourne **true** si le caractère *c* peut faire partie d'un identificateur Java et **false** sinon.
static boolean isJavaIdentifierStart(**char** c)	Retourne **true** si le caractère *c* peut être le premier caractère d'un identificateur Java et **false** sinon.

3. Number

3.1 Byte

La classe *Byte* possède trois constantes : *Byte.MAX_VALUE* qui vaut 2^7-1, *Byte.MIN_VALUE* qui vaut -2^7. et *Byte.SIZE* qui vaut 8 le nombre de bits de la représentation d'un *byte*.

3.2 Short

La classe *Short* possède trois constantes : *Short.MAX_VALUE* qui vaut 2^{15}-1, *Short.MIN_VALUE* qui vaut -2^{15}. et *Short.SIZE* qui vaut 16 le nombre de bits de la représentation d'un *short*.

3.3 Integer

La classe *Integer* possède trois constantes : *Integer.MAX_VALUE* qui vaut 2^{32}-1, *Integer.MIN_VALUE* qui vaut -2^{32} et *Integer.SIZE* qui vaut 32 le nombre de bits de la représentation d'un *int*.

`static int highestOneBit(int i)`	Retourne la valeur entière du bit 1 de plus fort poids de la représentation de *i*
`static int lowestOneBit(int i)`	Retourne la valeur entière du bit 1 de moins fort poids de la représentation de *i*
`static int numberOfLeadingZeros (int c)`	Retourne le nombre de 0 en début de la représentation de *i*.
`static int numberOfTrailingZeros (int c)`	Retourne le nombre de 0 en fin de la représentation de *i*.
`static int rotateLeft(int i, int d)`	Retourne la valeur de *i* à laquelle on a fait subir une rotation de distance *d* vers la gauche. Un bit peut aller occuper le bit de signe.
`static int rotateRight(int i, int d)`	Retourne la valeur de *i* à laquelle on a fait subir une rotation de distance *d* vers la droite. Un bit peut aller occuper le bit de signe.
`static int reverse(int i)`	Retourne la valeur obtenue en inversant l'ordre des bits de *i*.
`static int bitCount(int i)`	Retourne le nombre de bits à 1 dans la représentation de *i*.

3.4 Long

La classe *Long* possède trois constantes : *Long.MAX_VALUE* qui vaut 2^{63}-1, *Long.MIN_VALUE* qui vaut -2^{63} et *Long.SIZE* qui vaut 64 le nombre de bits de la représentation d'un *long*.

La classe *Long* a les mêmes méthodes que la classe *Integer*.

3.5 Float

La classe *Float* possède les constantes : *Float.MAX_VALUE* qui vaut $(2-2^{-23})2^{127}$, *Float.MIN_VALUE* qui vaut 2^{-149}, *Float.SIZE* qui vaut 32 le nombre de bits de la représentation d'un *Float*, *Float.NaN* une représentation de NotANumber, *Float.POSITIVE_INFINITY* qui représente $+\infty$ et *Float.NEGATIVE_INFINITY* qui représente $-\infty$.

Des méthodes permettent de «voir» un *int* comme un *float* et inversement.

3.6 Double

La classe *Double* possède les constantes : *Double.MAX_VALUE* qui vaut $(2-2^{-52})2^{1023}$, *Double.MIN_VALUE* qui vaut 2^{-1074}, *Double.SIZE* qui vaut 64 le nombre de bits de la représentation d'un *Double*, *Double.NaN* une représentation de NotANumber, *Double.POSITIVE_INFINITY* qui représente $+\infty$ et *Double.NEGATIVE_INFINITY* qui représente $-\infty$.

Des méthodes permettent de «voir» un *long* comme un *double* et inversement.