Chapitre 3

$$\text{Quelques} \left\{ \begin{array}{c} \text{paquetages} \\ \text{classes} \end{array} \right\} \text{prédéfinis}$$

$$\text{et quelques membres} \left\{ \begin{array}{l} \text{attributs} \\ \text{constructeurs} \\ \text{méthodes} \end{array} \right.$$

Quelques Paquetages prédéfinis

java.lang	classes de base
java.io	entrées sorties
java.util	les utilitaires
java.applet	les applets
java.awt	interface graphique (Abstract Window Toolkit)
java.awt.event	les événements
javax.swing	interface graphique

Quelques classes

java.lang.Object
- java.lang.Number
  - java.lang.Byte
  - java.lang.Short
  - java.lang.Integer
  - java.lang.Long
  - java.lang.Float
  - java.lang.Double
- java.lang.String
- java.util.StringTokenizer
- java.lang.StringBuffer
- java.lang.Math
- java.util.AbstractCollection
  - java.util.AbstractList
    - java.util.Vector
- java.util.Date
- java.util.Calendar
  - java.util.GregorianCalendar
- java.util.Random
- java.lang.Class
- java.text.Format
  - java.text.DateFormat
  - java.text.NumberFormat
    - java.text.DecimalFormat
- java.util.Dictionnary
  - java.util.Hashtable
- java.lang.System
- java.io.InputStream
- java.io.OutputStream
  - • - • java.io.PrintStream
- • - • java.lang.Exception
    - java.io.IOException

Documentations partielles

Listes non exhaustives d’attributs, constructeurs et méthodes.
Tous ces membres sont publics (sauf mention contraire).

La classe **Object**

package java.lang

`boolean` **`equals`**`(Object obj)`

Indicates whether some other object is *equal to* this one.

`String` **`toString`**`()`

Returns a string representation of the object.

`final Class` **`getClass`**`()`

Returns the runtime class of an object.

`protected Object` **`clone`**`()`

Creates and returns a copy of this object.

Programmation Java 2003/2004 - D.Pastre Chap.3.5

La classe **Integer**

package java.lang

`static int` **`MIN_VALUE`**

The smallest value of type `int`.

`static int` **`MAX_VALUE`**

The largest value of type `int`.

`Integer``(int value)`

Constructs a newly allocated `Integer` object that represents the primitive `int` argument

`Integer``(String s)`

Constructs a newly allocated `Integer` object that represents the value represented by the `String`.

Programmation Java 2003/2004 - D.Pastre Chap.3.6

`byte` **`byteValue`**`()`

returns the value of this `Integer` as a *`byte`*.

`short` **`shortValue`**`()`

.... `short`.

`int` **`intValue`**`()`

.... `int`.

`long` **`longValue`**`()`

.... `long`.

`float` **`floatValue`**`()`

.... `float`.

`double` **`doubleValue`**`()`

.... `double`.

`static int` **`parseInt`**`(String s)`

Parses the `String` argument as a signed decimal integer.

`static Integer` **`valueOf`**`(String s)`

Returns a new `Integer` object initialized to the value of the specified String

`boolean` **`equals`**`(Object obj)`

Compares this `Integer` to the specified `Object`.

`int` **`compareTo`**`(Integer anotherInteger)`

Compares two `Integer`*s* numerically.

Returns 0 if this `Integer` is numerically equal to the argument

<0	…	less than	…
>0	…	greater than	…

`int` **`compareTo`**`(Object o)`

Compares this `Integer` to another `Object`.

`String` **`toString`**`()`

Returns a `String` object representing this `Integer`'s value.

`static String` **`toString`**`(int i)`

Returns a new `String` object representing the specified `Integer`.

La classe **Double**

package java.lang

static int **MAX_VALUE**
    The largest positive finite value of type *double*.
static int **MIN_VALUE**
    The smallest positive value of type *double*.
static double **NaN**
    A Not-a-Number (NaN) value of type *double*.
static double **NEGATIVE_INFINITY**
    The negative infinity ot type *double*.
static double **POSITIVE_INFINITY**
    The positive infinity ot type *double*.

constructeurs analogues à ceux de la classe *Integer* à partir d'un *double* ou d'un *String*.

boolean **isNaN**()
    Returns true if this *Double* value is the special *NaN* value.
boolean **isInfinite**()
    Returns true if this *Double* value is infinitely large in magnitude.
static boolean **isNaN**(*double v*)
    Returns *true* if the specified number is the special *NaN* value.
boolean **isInfinite**(*double v*)
    Returns *true* if the specified number is infinitely large in magnitude.

byteValue()
shortValue()
intValue()
longValue()   analogues aux fonctions de *Integer*
floatValue()
doubleValue()

static Double **valueOf**(*String s*)
Returns a new *Double* object initialized to the value represented by the specified *String* .
static double **parseDouble**(*String s*)
Returns a new *double* initialized to the value represented by the specified *String* .

boolean **equals**(*Object obj*)
Compares this *Double* to another *Object*.
int **compareTo**(*Double anotherDouble*)
Compares two *Doubles* numerically.
int **compareTo**(*Object o*)
Compares this *Double* to another *Object*.

toString()
toString(*double d*) analogues aux fonctions de *Integer*

La classe **String**

package java.lang
boolean **equals**(*Object anObject*)
compares this *String* to the specified *Object*
int **compareTo**(*String anotherString*) *ou* (*Object* o)
compares two *String*s lexicograpphically, returns 0, <0, >0
int **length**()
returns the length of this *String*
static String **valueOf**(*int i*)
returns the *String* representation of the *int* argument
et analogues pour les types élémentaires *long, float, double, char,boolean*
char **charAt**(*int index*)
returns the character at the specified index

int **indexOf**(*int ch*) *int* **indexOf**(*int ch, int fromIndex*)
int **indexOf**(*String str*) *int* **indexOf**(*String str, int fromIndex*)

String **concat** (*String str*) *ou* opérateur + infixe

concatenates the specified string to the end of this *String*

String **replace**(*char oldChar, char newChar*)

returns a new *String* resulting from replacing all occurrences of *oldChar* in the *String* with *newChar*

String **toLowerCase**()

converts all the characters in this *String* to lower case

String **toUpperCase**()

converts all the characters int this *String* to upper case

String **trim**()

removes white space from both ends of this *String*

Ces méthodes ne modifient pas les chaines mais en créent de nouvelles. En particulier, dans la concaténation s1.concat(s2) *ou* s1+s2, s2 *est ajouté à une* ***copie*** *de* s1

La classe **StringBuffer**

package java.lang

StringBuffer()
StringBuffer(String str)

StringBuffer **append**(String str)

contrairement aux **String**, il y a seulement une *concaténation*, et non *création* d'une nouvelle chaine

int **length**()
char **charAt**(int index)
void **setCharAt**(int index, char ch)
StringBuffer **reverse**()

les chaines sont modifiées, il n'y a pas de nouvelle création

String **toString**()

pas de méthodes **equals** ni **compareTo** ni **indexOf**

Exemple

```
public static String toString(int[] tab) {
   String chaine = "";
   for (int i=0;i<tab.length;i++) chaine +=" "+tab[i];
   return chaine;
    }
```

Meilleur !

```
public static String toString(int[] tab) {
   StringBuffer chaine = new StringBuffer();
   for (int i=0;i<tab.length;i++) chaine.append(" "+tab[i]);
   return chaine.toString();
   }
```

La classe **Math**

package java.lang

static double { E, PI

static double ou *float* ou *int* ou *long* {

	round
abs	floor
sqrt	ceil
pow	rint
max	sin
min	cos
	tan
log	
exp	asin
	acos
random	atan

La classe **Vector**

package java.util

Vector()
Constructs an empty vector.

void **addElement(***Object obj***)**
Adds the specified component to the end of this *Vector*, increasing its size by one.
boolean **removeElement(*Object obj*)**
Removes the first occurrence of the argument from this *Vector*.
Object elementAt(*int index***)**
Returns the component at the specified index.
void **setElementAt(***Object obj, int index***)**
Sets the component at the specified index of this *Vector* to be the specified object

void **removeElementAt(***int index***)**
Deletes the component at the specified index.
int **size()**
Returns the number of components in this *Vector*.
boolean **isEmpty()**
Tests if this *Vector* has no components.
boolean **contains(Object elem)**
Tests if the specified object is a component in this *Vector*.
int **indexOf(Object elem)**
Searches for the first occurence of the given argument, testing for equality using the *equals* method.
int **indexOf(***Object elem, int index***)**
Searches for the first occurence of the given argument, beginning the search at index, and testing for equality using the *equals* method.

Object **firstElement()**
Returns the first component of this *Vector*.
Object **lastElement()**
Returns: the last component of this *Vector*, i.e., the component at index size() - 1.
void **insertElementAt(***Object obj, int index***)**
Inserts the specified object as a component in this *Vector* at the specified index.
String **toString()**
Returns a string representation of this *Vector*.

+ (voir chapitre sur les sous-classes)
Enumeration **elements**();
Returns an enumeration of the components of this *Vector*

La classe **Random**

package java.lang

Random()
creates a new random number generator.

double **nextInt**()
returns the next pseudorandom, uniformly distributed *int* value, from this random number generator's sequence.
double **nextInt**(*int* n)
… between *0* (inclusive) and the specified value (exclusive) …
double **nextDouble**()
… *double* value, between *0.0* et *1.0* …

La classe **Date**

package java.util

Date()

Allocates a *Date* object and initializes it so that it represents the time at which it was allocated, measured to the nearest millisecond.

Date(long date)

Allocates a *Date* object and initializes it to represent the specific number of milliseconds since the standard base time known as the "epoch" , namely January 1, 1970, 00:00:00 GMT.

long **getTime**()

Returns the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this *Date* object.

void **setTime**(*long time*)

Sets this *Date* object to represent a point in time that is *time* milliseconds after January 1, 1970 00:00:00 GMT.

Programmation Java 2003/2004 - D.Pastre Chap.3.21

String **toString**()

Converts this *Date* object to a *String* of the form:

dow mon dd hh:mm:ss zzz yyyy

boolean **after**(Date when)

Tests if this *Date* is after the specified *Date*.

boolean **before**(*Date when*)

………………… before ....

int **compareTo**(*Date anotherDate*)

Compares two *Date*s for ordering. Returns 0, <0, >0.

boolean **equals**(*Object obj*)

Compares two *Date*s for equality.

Programmation Java 2003/2004 - D.Pastre Chap.3.22

La classe **GregorianCalendar**

package java.util

GregorianCalendar()
Constructs a default *GregorianCalendar* using the current time in the default time zone with the default locale.
GregorianCalendar(*int year, int month, int date*)
GregorianCalendar(*int year, int month, int date, int hour, int minute, int second*)
Constructs a *GregorianCalendar* with the given date set in …

boolean **equals**(*Object obj*)
Compares this *GregorianCalendar* to an object reference.
void **add**(*int field,int amount*)
Adds the specified (signed) amount of time to the given time field, based on the calendar's rules.

La classe **Calendar**

public abstract class Calendar package java.util

static int **YEAR**
Field number for *get* and *set* indicating the year.
static int **MONTH**
Field number for *get* and *set* indicating the month.
static int **JANUARY**
Value of the MONTH field indicating the first month of the year.
static int **DAY_OF_YEAR**
static int **DAY_OF_MONTH**
static in **DAY_OF_WEEK**
static int **MONDAY**
Value of the *DAY_OF_WEEK* field indicating Monday.
static int **HOUR_OF_DAY**
Field number for *get* and *set* indicating the hour of the day.

Date **getTime**()
  Gets this *Calendar*'s current time.
void **setTime**(*Date date*)
  Sets this *Calendar*'s current time with the given *Date*.
String **toString**()
  Returns a *String* representation of this *Calendar*.

void **add**(*int field,int amount*)
  Adds the specified (signed) amount of time to the given time field, based on the calendar's rules.

La classe **Class**

package java.lang

public String **getName**()
  Returns the fully-quantified name of the entity (class, interface, array class, primitive type, or void) represented by this *Class* object, as a *String*.
public Package **getPackage**()
  Gets the package of this class.
public String **toString**()
  Converts the *Object* to a *String*.
public boolean **isInstance**(*Object* o)
  Determines if the specified *Object* is assignment-compatible with the object represented by this *Class*.

Exemples

```
Vector v = new Vector(); v.addElement(new Integer(5));
Class C = v.getClass(); System.out.println(C);
String nom = C.getName(); System.out.println(nom);
Object o = v.elementAt(0); nom = o.getClass().getName();
System.out.println(nom);
Integer I = (Integer) o; nom = I.getClass().getName();
System.out.println(nom);
System.out.println(C.isInstance(v)+" "+C.isInstance(o)+
                                   " "+C.isInstance(I));
```

affiche

```
        class java.util.Vector
        java.util.Vector
        java.lang.Integer
        java.lang.Integer
        true false false
```

l'opérateur **instanceof**

Son rôle est de contrôler l'héritage
(o **instanceof** C) est vrai si *o* , qui est déjà une instance d'une super-classe de *C*, est une instance de la classe *C*

Exemple
Soient v, o, I comme précédemment

```
System.out.println(v instanceof Vector);
System.out.println(o instanceof Object+" "+o instanceof Integer
                                      +" "+o instanceof Double);
System.out.println(I instanceof Object+" "+I instanceof Integer);
// System.out.println(I instanceof Double); erreur
```

affiche

```
        true
        true true false
        true true
```

La classe **DecimalFormat**

package java.text

DecimalFormat(*String pattern*)
Create a *DecimalFormat* from the given pattern

La classe **NumberFormat**

package java.text

public String **format**(*double number*)
Specialization of format.
public String **format**(*long number*)
Specialization of format.

Exemple

```
d=4.0/3;
DecimalFormat df = new DecimalFormat("0.00");
System.out.println(d+"\n"+df.format(d)+"   "+df.format(d/10));
DecimalFormat df1 = new DecimalFormat("0000.0000");
System.out.println(df1.format(d)+"   "+df1.format(d/10));
DecimalFormat df2 = new DecimalFormat(".0000");
System.out.println(df2.format(d)+"   "+df2.format(d/10));
```

affiche

```
1.3333333333333333
1.33   0.13
0001.3333   0000.1333
1.3333   .1333
```

La classe **Hashtable**

package java.util

Pour mémoriser des listes d'objets et les retrouver facilement grâce à une clef

Exemple

This example creates a hashtable of numbers.
It uses the names of the numbers as keys:

```
Hashtable numbers = new Hashtable();
    numbers.put("one", new Integer(1));
    numbers.put("two", new Integer(2));
    numbers.put("three", new Integer(3));
```

To retrieve a number, use the following code:

```
Integer n = (Integer)numbers.get("two");
    if (n != null)  System.out.println("two = " + n);
```

Hashtable() package java.util
  Constructs a new, empty *Hashtable*.

Object **put**(*Object key, Object value*)
  Maps the specified key to the specified value in this *Hashtable*.

Object **get**(*Object key*)
  Returns the value to which the specified key is mapped in this *Hashtable*.

Object **remove**(*Object key*)
  Removes the key (and its corresponding value) from this *Hashtable*.

int **size**()
  Returns the number of keys in this *Hashtable*.

String **toString**()
  Returns a *String* representation of this *Hashtable* object in the form of a set of entries, enclosed in braces and separated by the ASCII characters ", " (comma and space).

boolean **isEmpty**()
Tests if this *Hashtable* maps no keys to values.
boolean **contains**(*Object value*)
Tests if some key maps into the specified value in this *Hashtable*.
boolean **containsKey**(*Object key*)
Tests if the specified object is a key in this *Hashtable*.

String **toString**()
Returns a *String* representation of this *Hashtable* object in the form of a set of entries, enclosed in braces and separated by the ASCII characters ", " (comma and space).

+ (voir chapitre sur les sous-classes)
Enumeration **elements**();
Returns an enumeration of the values of this *Hashtable*.
Enumeration **keys**();
Returns an enumeration of the keys in this *Hashtable*

La classe **StringTokenizer**

Allows an application to break a string into tokens. The set of delimiters (the characters that separate tokens) may be specified either at creation time or on a per-token basis.
StringTokenizer (*String str*)
Constructs a string tokenizer for the specified string
StringTokenizer (*String str, string delim*)
Constructs a string tokenizer for the specified string. The character in the delim argument are the delimiters for separating tokens.
boolean **hasMoreElements**() ou *boolean* **hasMoreTokens**()
Test if there are more tokens available for this tokenizer's string.
String **nextElement**() ou *Object* **nextElement**()
Returns the next token from this string tokenizer

Exemple d'utilisation

```
String s = "Le corbeau et le renard.";
StringTokenizer st = new StringTokenizer(s);
while (st.hasMoreTokens()) System.out.println(st.nextToken());
```

affiche

```
Le
corbeau
et
le
renard.
```

La classe **System**

package java.lang

static PrintStream **out**
The "standard" output stream
static InputStream **in**
The "standard" input stream

La classe **PrintStream**

package java.io

void **print**(*String s*)
Print a *String*.

void **print**(*int i*)
Print an *integer*.

void **print**(*double d*)
Print a *double*.

La classe **InputStream**

package java.io

int **read**()
Reads the next byte of data from the input stream.