

FAQ des TP Java

A. Lebret, 2025

Plan

- Equivalences entre C et Java ? ✓
- Où récupérer la documentation officielle ? ✓
- Les outils JetBrains à l'ENSICAEN ?
- Comment documenter une classe ? ✓
- **monAttribut** ou **_monAttribut** ?
- Quel est le lien entre **equals()** et **hashCode()** ?
- Différence entre l'opérateur **==** et la méthode **equals()** ?
- Comment récupérer les informations du système ?
- Comment choisir le bon flux de données ?
- Qu'est-ce qu'une interface ?
- Qu'est-ce qu'un générique ?
- **Object** ou générique ?
- Pourquoi déclarer un attribut à l'aide de son interface générique est-il préférable ?
- Comment utiliser les lambdas ?
- Comment tester les exceptions avec JUnit ?
- Comment implémenter et synchroniser les *threads* ?
- Comment manipuler les expressions régulières ?

Équivalences entre C et Java ?

Equivalences C et Java ?

Langage	C	Java
Compilateur	<code>gcc [options] prog.c</code>	<code>javac [options] MaClasse.java</code>
Débogueur	<code>gdb [options] ./prog</code>	<code>jdb [options] MaClasse</code>
Documentation	<code>doxygen [DOXYFILE]</code>	<code>javadoc MaClasse.java</code>
Construction	<code>make</code>	<code>mvn, gradle, ant, (make)</code>
Test	<i>Minunit, CUnit</i>	<i>JUnit (v. 4 ou 5)</i>
Lancement	<code>./prog</code>	<code>java Maclasse</code>

Apache
Google

The diagram shows two purple arrows. One arrow starts from the word 'Apache' and points to the 'mvn' box. The other arrow starts from the word 'Google' and points to the 'gradle' box. The words 'mvn' and 'gradle' are each enclosed in a purple rounded rectangle.

Exemple de Makefile

```
JAVAC=javac
JFLAGS=-g
RM=rm

all: Telegramme.class EcritDouble.class Troncature.class

Telegramme.class: Telegramme.java
    $(JAVAC) $(JFLAGS) $^

EcritDouble.class: EcritDouble.java
    $(JAVAC) $(JFLAGS) $^

Troncature.class: Troncature.java
    $(JAVAC) $(JFLAGS) $^

doc:
    javadoc -d doc *.java

clean:
    $(RM) *.class

.PHONY: all doc clean
```

**Où récupérer la documentation
officielle ?**

Documentation Java officielle

<https://docs.oracle.com/javase/8/docs/api/index.html>

Java™ Platform
Standard Ed. 8

All Classes All Profiles

Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event

All Classes

AbstractAction
AbstractAnnotationValueVisitor6
AbstractAnnotationValueVisitor7
AbstractAnnotationValueVisitor8
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractChronology
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeContext
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractElementVisitor6
AbstractElementVisitor7
AbstractElementVisitor8
AbstractExecutorService
AbstractInterruptibleChannel
AbstractLayoutCache
AbstractLayoutCache.NodeDimension
AbstractList
AbstractListModel
AbstractMap
AbstractMap.SimpleEntry
AbstractMap.SimpleImmutableEntry
AbstractMarshallerImpl

Java™ Platform
Standard Ed. 8

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV NEXT FRAMES NO FRAMES

Java™ Platform, Standard Edition 8 API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Profiles

- compact1
- compact2
- compact3

Packages

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events.

N° de version

Les outils Jetbrains à l'ENSICAEN ?

Les outils JetBrains à l'ENSICAEN ?

- Lancer CLion (C/C++) ou IntelliJ (Java)
- Choisir « **Licence Server** »
- Entrer l'URL : **<http://jetbrains.ecole.ensicaen.fr:8080>**

Remarque : Les produits *Jetbrains* sont disponibles en version « communautaire » (limitée) ou « professionnelle » (avec votre courriel ENSICAEN)

Comment documenter une classe ?

Documenter une classe

```
/*  
 * ENSICAEN  
 * 6 Boulevard Maréchal Juin  
 * F-14050 Caen Cedex  
 *  
 * This file is owned by ENSICAEN students.  
 * No portion of this document may be reproduced, copied  
 * or revised without written permission of the authors.  
 */  
package fr.ensicaen.tp2.ex1;  
  
/**  
 * A class to manipulate various documents. Documents are  
 * represented by an identifier (a unique integer) and a title.  
 *  
 * @author Jean Saigne (jean.saigne at ensicaen.fr)  
 * @version 1.0  
 */  
public class Document {
```

Documenter une classe

```
/** Identifier of the document. It should be unique ! */  
private int identifier;
```

```
/** Title of the document. It should not be an empty string */  
private String title;
```

```
/**  
 * Creates a new document with given identifier and title.  
 *  
 * @param identifier The unique identifier of the document  
 * @param title The non empty title of the document  
 */
```

```
public Document(int anIdentifier, String aTitle) {
```

```
    ...
```

```
}
```

```
}
```

monAttribut ou _monAttribut ?

Nommage des attributs

Critère	String monAttribut	String _monAttribut
Lisibilité	Très bonne, naturelle, simple	Moins naturelle à la lecture
Différenciation attribut / paramètre	Nécessite this. pour éviter la confusion (ex. : this.monAttribut)	Claire immédiatement sans utiliser this
Respect des standards	Oui, recommandé officiellement par Oracle	Non, rarement utilisé, déconseillé par les conventions Java officielles
Facilité de réusinage (IDE)	Bonne	Bonne
Compatibilité avec bibliothèques externes	Parfaite compatibilité, usage universel	Moins courant, confusion possible avec bibliothèques utilisant cette notation

Quels est le lien entre equals() et hashCode() ?

Contrat de equals()

@Override

```
public boolean equals(Object o) {  
    if (o == this) {                // étape 1  
        return true;  
    }  
    if (!(o instanceof MaClasse)) { // étape 2  
        return false;  
    }  
    Maclasse mc = (MaClasse)o;      // étape 3  
    boolean isEqual = mc.attribut1 == attribut1  
        && mc.attribut2 == attribut2  
        && ...;  
    return isEqual;  
}
```

Rôle de hashCode()

- **hashCode()** retourne une valeur entière **unique** associée à un objet
- Par défaut, **hashCode()** dans **Object** retourne l'**adresse mémoire** à laquelle est stockée l'instance

HashMap et HashSet utilisent les « hashcodes » pour classer / localiser les objets (cf. tables de hachage - association clés/valeurs)

Lien entre equals() et hashCode()

- 2 objets égaux au sens de **equals()** doivent retourner le même **hashCode**
- Surcharger **equals()** implique de surcharger **hashCode()**

Le non-respect de cette règle risque d'exposer à des erreurs très difficiles à détecter !!!!

Comment surcharger hashCode() ?

- Supposons que **equals()** teste l'égalité entre plusieurs attributs de la classe
- On choisit deux nombres premiers pas trop petits (**17** et **31** par exemple) et on initialise la valeur de retour à **17**
- Pour chacun des attributs **attr** pris en compte par **equals()** on construit l'entier **hash** :
 - Si **boolean** ALORS `hash = 1` Si **true**, `0` SINON
 - Si **byte**, **short**, **int**, **char** ALORS `hash = (int)attr`
 - Si **long** ALORS `hash=(int)(attr^(attr >>> 32))`
 - Si **float** ALORS `hash=Float.floatToIntBits(attr)`
 - Si **double** ALORS `hash=Double.doubleToLongBits(attr)` et on prend le code de hachage du **long** que l'on récupère

Comment surcharger hashCode() ?

- Si **attr==null** ALORS `hash=0`
- Si **attr** est un objet non nul ALORS `hash=attr.hashCode()`
- Si **attr** est un tableau ALORS chacun des éléments du tableau est traité comme un attribut à part entière
- Pour chacun de ces attributs, on met à jour la valeur de retour :
`result = 31 * result + hash`

Différence entre l'opérateur == et la méthode equals() ?

Ah les String(s) : javac optimise

```
String abcd = "Bonjour vous";  
String wxyz = "Bonjour vous";
```

javac place les 2 chaînes à la même adresse

```
if (abcd == wxyz) {  
    System.out.println("[test1/==] Chaînes identiques");  
} else {  
    System.out.println("[test1/==] Chaînes différentes");  
}
```

```
if (abcd.equals(wxyz)) {  
    System.out.println("[test1/equals] Chaînes identiques");  
} else {  
    System.out.println("[test1/equals] Chaînes différentes");  
}
```

Ah les String(s) ! javac optimise en réaffectant

`String abcd = "Ragna", wxyz = "Roc";` → *javac considère 2 objets distincts*

`abcd = "Bonjour vous";`
`wxyz = "Bonjour vous";` } → *javac réaffecte les 2 chaînes à la même adresse*

```
if (abcd == wxyz) {  
    System.out.println("[==] Chaînes identiques");  
} else {  
    System.out.println("[==] Chaînes différentes");  
}
```


```
if (abcd.equals(wxyz)) {  
    System.out.println("[equals] Chaînes identiques");  
} else {  
    System.out.println("[equals] Chaînes différentes");  
}
```

Ah les String(s) ! javac n'optimise plus

```
int i = 0;
String ab = "Bonjour";
StringTokenizer st = new StringTokenizer("Bonjour vous", " ");
String[] mots = new String[st.countTokens()];

while (st.hasMoreTokens()) {
    String mot = st.nextToken();
    mots[i] = mot;
    i++;
}
if ((mots[0] == ab)) {
    System.out.println("[test3/==] Chaînes identiques");
} else {
    System.out.println("[test3/==] Chaînes différentes");
}
if ((mots[0].equals(ab))) {
    System.out.println("[test3>equals] Chaînes identiques");
} else {
    System.out.println("[test3>equals] Chaînes différentes");
}
```

séparateur



Comment récupérer les informations du système ?

Chemins en « dur » ou pas ?

- Évitez les chemins en « dur » dans le code :

```
public FilteredDirectory(String folder, String extension) {  
    _folder = "../.." + folder;  
    _extension = extension;  
}
```

Séparateurs de chemin

- Unix ou Mac OS X : « / »
- Ms-Windows : « \ »

1. `File.separator`

```
String separateur = File.separator;
```

2. `File.separatorChar`

```
char separateur = File.separatorChar;
```

3. Paquetage `java.nio`

```
String separateur =  
    FileSystems.getDefault().getSeparator();
```

Récupération du dossier courant

1. `System.getProperty()`

```
String dossierCourant = System.getProperty("user.dir");
```

2. `File.getAbsolutePath()`

```
String dossierCourant = new File("").getAbsolutePath();
```

3. Paquetage `java.nio`

```
String dossierCourant = FileSystems.getDefault()  
    .getPath("").toAbsolutePath().toString();
```

4. Paquetage `java.nio`

```
String dossierCourant =  
    Paths.get("").toAbsolutePath().toString();
```

Comment choisir un flux de données ?

Comment choisir un flux de données ?

Classes abstraites d'entrées-sorties

Flux de données en entrée

Paquetage pour les E/S (voir aussi `javax.nio`)

`java.io`

Class Reader

`java.lang.Object`
`java.io.Reader`

All Implemented Interfaces:

`Closeable`, `AutoCloseable`, `Readable`

Les classes filles dans lesquelles chercher

Direct Known Subclasses:

`BufferedReader`, `CharArrayReader`, `FilterReader`, `InputStreamReader`, `PipedReader`, `StringReader`

```
public abstract class Reader
extends Object
implements Readable, Closeable
```

Abstract class for reading character streams. The only methods that a subclass must implement are `read(char[], int, int)` and `close()`. Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:

JDK1.1

See Also:

`BufferedReader`, `LineNumberReader`, `CharArrayReader`, `InputStreamReader`, `FileReader`, `FilterReader`, `PushbackReader`, `PipedReader`, `StringReader`, `Writer`

Flux de caractères d'entrée

Flux de caractères d'entrée (étendant la classe abstraite **Reader**) :

- **CharArrayReader** : lecture dans un tableau de caractères
- **BufferedReader** : lecture avec un tampon
- **LineNumberReader** : permet de connaître le numéro de ligne
- **InputStreamReader** : lecture dans un flux d'octets
- **FileReader** : lecture dans un fichier
- **PipedReader** : lecture dans un tube
- **StringReader** : lecture depuis une chaîne de caractères
- **FilterReader** : ajoute un traitement sur le flux d'entrée
- **PushbackReader** : permet de remettre des caractères dans le flux

Flux de données de sortie

java.io

Class Writer

java.lang.Object
java.io.Writer

All Implemented Interfaces:

Closeable, Flushable, Appendable, AutoCloseable

Direct Known Subclasses:

BufferedWriter, CharArrayWriter, FilterWriter, OutputStreamWriter, PipedWriter, PrintWriter, StringWriter

```
public abstract class Writer  
extends Object  
implements Appendable, Closeable, Flushable
```

Abstract class for writing to character streams. The only methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:

JDK1.1

See Also:

Writer, BufferedWriter, CharArrayWriter, FilterWriter, OutputStreamWriter, FileWriter, PipedWriter, PrintWriter, StringWriter, Reader

Comment choisir un flux de données ?

Associer les flux

Associer les flux suivant les circonstances

Vous souhaitez lire les caractères d'un chaîne de caractères :

```
Reader r1;  
int c;  
  
r1 = new StringReader("Premier essai");  
while ((c = r1.read()) != -1) {  
    System.out.println((char) c);  
}
```

Et en utilisant un tampon :

```
Reader r1, r2;  
  
r1 = new StringReader("Deuxième essai");  
r2 = new BufferedReader(r1);  
System.out.println(r2.readLine());
```

Comment choisir un flux de données ?

Flux de données et fichier

Lire des chaînes de caractères depuis un fichier

The screenshot shows the Java Platform Standard Ed. 8 documentation for the `FileReader` class. The left sidebar lists various Java packages, including `java.io`. The main content area is divided into several sections: **Field Summary**, **Constructor Summary**, **Method Summary**, and **Constructor Detail**. Handwritten purple annotations are present:

- 1. Allez voir comment construire l'objet (constructeur)**: An arrow points from this text to the **Constructors** section.
- 2. Consulter les méthodes disponibles**: An arrow points from this text to the **Method Summary** section.
- 3. Si elles ne conviennent pas : chercher une classe à « emboîter » pour faciliter la lecture**: This text is located near the **Method Summary** section.
- 4. Aller voir la documentation de la classe File**: This text is located near the **Constructor Summary** section.

The **Constructors** section lists three constructors:

- `FileReader(File file)`**: Creates a new `FileReader`, given the `File` to read from.
- `FileReader(FileDescriptor fd)`**: Creates a new `FileReader`, given the `FileDescriptor` to read from.
- `FileReader(String fileName)`**: Creates a new `FileReader`, given the name of the file to read from.

The **Method Summary** section lists methods inherited from `java.io.InputStreamReader` and `java.io.Reader`:

- Methods inherited from class `java.io.InputStreamReader`**: `close`, `getEncoding`, `read`, `read`, `ready`.
- Methods inherited from class `java.io.Reader`**: `mark`, `markSupported`, `read`, `read`, `reset`, `skip`.
- Methods inherited from class `java.lang.Object`**: `clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`.

Classes à associer

Flux de caractères d'entrée (étendant la classe abstraite **Reader**) :

- **CharArrayReader** : lecture dans un tableau de caractères
- **BufferedReader** : lecture avec un tampon
- **LineNumberReader** : permet de connaître le numéro de ligne
- **InputStreamReader** : lecture dans un flux d'octets
- **FileReader** : lecture dans un fichier
- **PipedReader** : lecture dans un tube
- **StringReader** : lecture depuis une chaîne de caractères
- **FilterReader** : ajoute un traitement sur le flux d'entrée
- **PushbackReader** : permet de remettre des caractères dans le flux

Lire des chaînes de caractères depuis un fichier

Constructors

Constructor and Description

BufferedReader(**Reader** in)

Creates a buffering character-input stream that uses a default-sized input buffer.

BufferedReader(**Reader** in, int sz)

Creates a buffering character-input stream that uses an input buffer of the specified size.

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type

Method and Description

void

close()

Closes the stream and releases any system resources associated with it.

Stream<**String**>

lines()

Returns a **Stream**, the elements of which are lines read from this **BufferedReader**.

void

mark(int readAheadLimit)

Marks the present position in the stream.

boolean

markSupported()

Tells whether this stream supports the **mark()** operation, which it does.

int

read()

Reads a single character.

int

read(char[] cbuf, int off, int len)

Reads characters into a portion of an array.

String

readLine()

Reads a line of text.

boolean

ready()

Tells whether this stream is ready to be read.

Lire des chaînes de caractères depuis un fichier

readLine

```
public String readLine()  
    throws IOException
```

Reads a line of text. A line is considered to be terminated by any one of a line feed ('\n'), a carriage return ('\r'), or a carriage return followed immediately by a linefeed.

Returns:

A String containing the contents of the line, not including any line-termination characters, or null if the end of the stream has been reached

Throws:

IOException - If an I/O error occurs

See Also:

Files.readAllLines(java.nio.file.Path, java.nio.charset.Charset)

la méthode devra être appelée depuis
avec try ... catch

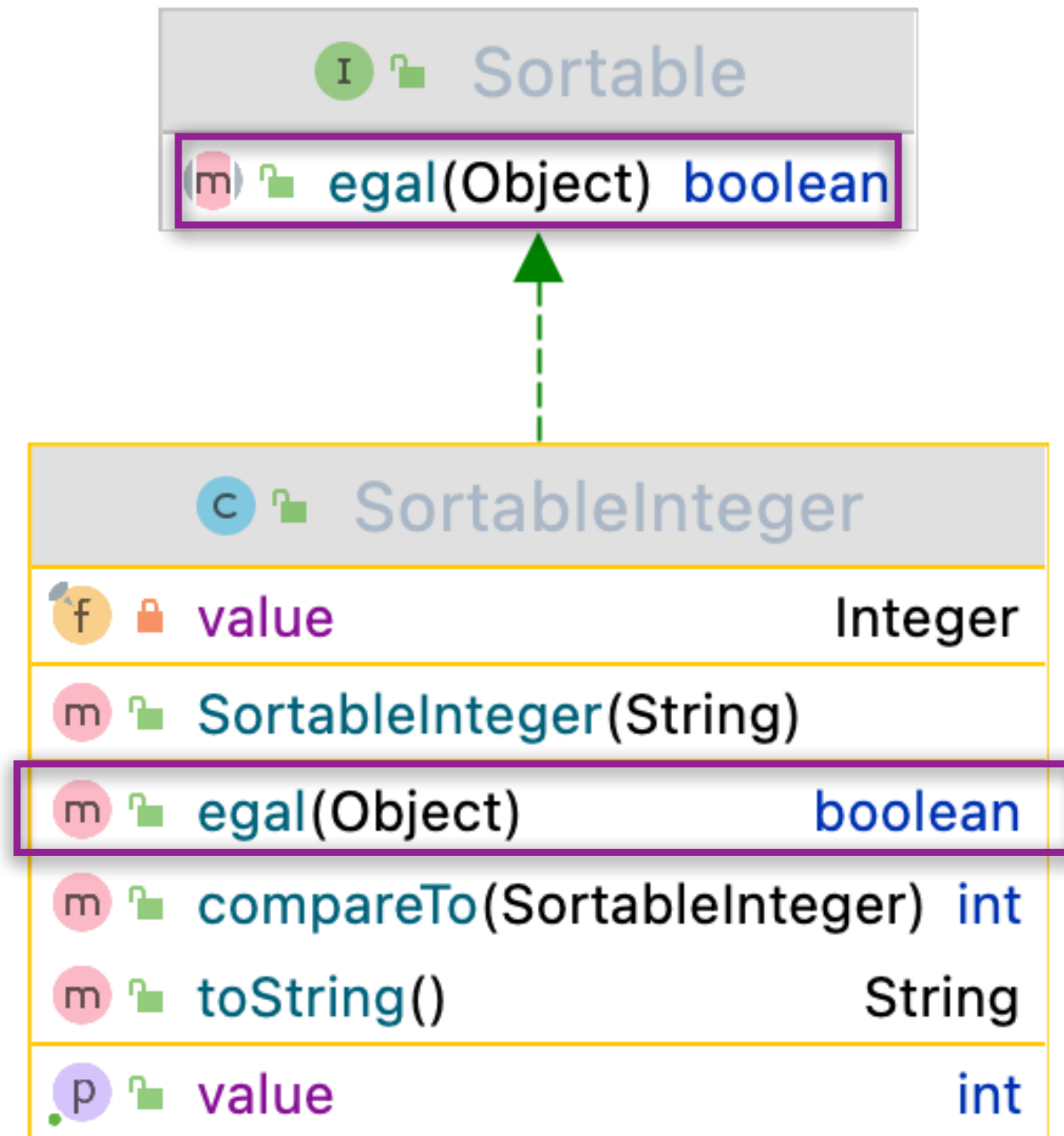


Qu'est-ce qu'une interface ?

Interface Java

- Une interface fournit un ou plusieurs **contrats** par l'intermédiaire de méthodes à implémenter
- Les classes qui implémentent l'interface devront **respecter le contrat** en définissant la ou les méthodes

Interface Java



Qu'est-ce qu'un générique ?

Génériques

Les génériques (version 5 et supérieure) :

- permettent d'**enrichir le polymorphisme**
- renforcent le typage statique
- évitent l'utilisation du transtypage (*cast*)

Les génériques sont mis en oeuvre dans les **collections**

Génériques

- Sans générique, c'est le programmeur qui doit être certain de gérer le bon type
- Supposons une liste d'entiers :

liste peut contenir n'importe quoi

```
List liste = new ArrayList();  
liste.add(1);  
Integer entier = (Integer) liste.get(0);
```

```
liste.add(1d);  
Integer entier = (Integer) liste.get(0);
```

compile, mais lève ClassCastException

Génériques

- Avec générique :

```
List<Integer> liste = new ArrayList<>();  
liste.add(1d); // ne compile pas
```

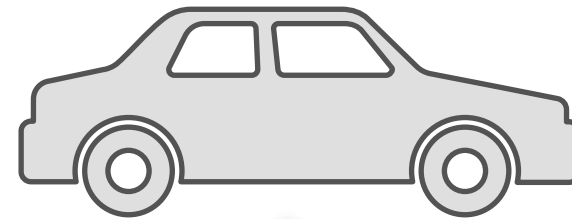
liste ne peut contenir que des « Integer »

Mieux : la conversion n'est plus nécessaire :

```
liste.add(1);  
Integer entier = list.get(0); // Plus besoin du transtypage
```

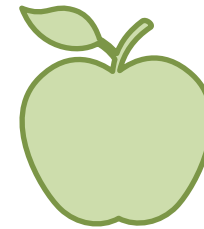
Object ou générique ?

Objects ou génériques ?



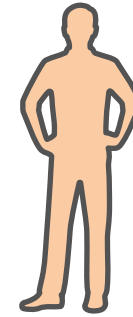
```
public class TamponVoitures {  
    Voiture[] voitures;  
    int nbVoitures;  
  
    public TamponVoitures(int c) {  
        voitures = new Voiture[c];  
    }  
  
    synchronized void push(Voiture v) {  
        // ...  
    }  
}
```

Objects ou génériques ?



```
public class TamponPommes {  
    Pomme[] pommes;  
    int nbPommes;  
  
    public TamponPommes(int c) {  
        pommes = new Pomme[c];  
    }  
  
    synchronized void push(Pomme p) {  
        // ...  
    }  
}
```

Objects ou génériques ?



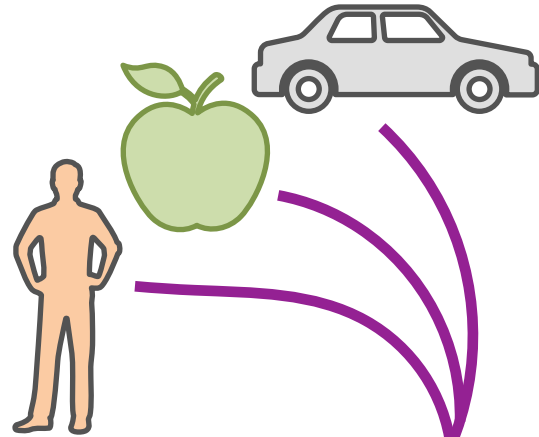
```
public class TamponEtudiants {  
    Etudiant[] etudiants;  
    int nbEtudiants;  
  
    public TamponEtudiants(int c) {  
        etudiants = new Etudiant[c];  
    }  
  
    synchronized void push(Etudiant e) {  
        // ...  
    }  
}
```

Objects ou génériques ?

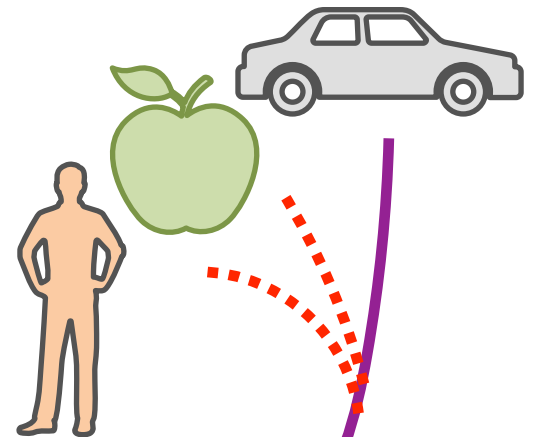
```
public class TamponObjets {  
    Object[] elem;  
    int nbElements;  
  
    public TamponObjets(int c) {  
        elem = new Object[c];  
    }  
  
    synchronized void push(Object e) {  
        // ...  
    }  
}
```

```
public class Tampon<T> {  
    T[] elem;  
    int nbElements;  
  
    public Tampon(int c) {  
        elem = (T[]) new Object[c];  
    }  
  
    synchronized void push(T e) {  
        // ...  
    }  
}
```

Objects ou génériques ?



```
public class TamponObjets {  
    Object[] elem;  
    int nbElements;  
  
    public TamponObjets(int c) {  
        elem = new Object[c];  
    }  
  
    synchronized void push(Object e) {  
        // ...  
    }  
}
```



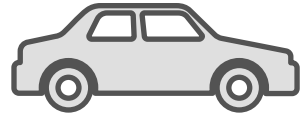
```
public class Tampon<T> {  
    T[] elem;  
    int nbElements;  
  
    public Tampon(int c) {  
        elem = (T[]) new Object[c];  
    }  
  
    synchronized void push(T e) {  
        // ...  
    }  
}
```

À la compilation

Objects ou génériques ?



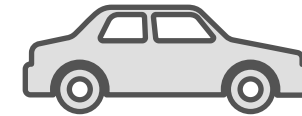
```
public class TamponObjets {  
    Object[] elem;  
    int nbElements;  
  
    public TamponObjets(int c) {  
        elem = new Object[c];  
    }  
  
    synchronized void push(Object e) {  
        // ...  
    }  
}
```



```
public class Tampon<T> {  
    T[] elem;  
    int nbElements;  
  
    public Tampon(int c) {  
        elem = (T[]) new Object[c];  
    }  
  
    synchronized void push(T e) {  
        // ...  
    }  
}
```

A l'exécution

Objects ou génériques ?



Tampon<Voiture> t;

```
public class Tampon<T> {  
    T[] elem;  
    int nbElements;  
  
    public Tampon(int c) {  
        elem = (T[]) new Object[c];  
    }  
  
    synchronized void push(T e) {  
        // ...  
    }  
}
```

Pourquoi déclarer un attribut à l'aide de son interface générique est-il préférable ?

Interface générique ou implémentation ?

```
public class AClass {  
    private ArrayList<String> list;  
  
    public AClass() {  
        list = new ArrayList<String>();  
    }  
}
```

```
public class ABetterClass {  
    private List<String> list;  
  
    public ABetterClass() {  
        list = new ArrayList<String>();  
    }  
}
```

Critères de choix d'une implémentation

Critère	Exemple concret d'utilisation	Structure adaptée
Accès rapide par index	Accès immédiat à des éléments fréquents (ex : liste d'étudiants)	ArrayList
Insertions/suppressions fréquentes	Historique de navigation, modifications fréquentes dans une liste	LinkedList ou DoubleLinkedList
Accès rapide par clé	Recherche par ID utilisateur	HashMap
Données triées automatiquement	Liste de scores triée par ordre alphabét. ou numérique	TreeMap ou TreeSet
Gestion de doublons	Autoriser/interdire des doublons	List (autorisé), Set (interdit)

Interface générique

- Soit une bibliothèque de gestion de données qui nous permet de stocker et de manipuler des informations dans différentes structures de données (par exemple **HashMap**, **ArrayList** ou **LinkedList**)
- Nous voulons écrire un programme qui utilise ces classes pour stocker et afficher des informations
- Nous pouvons créer une interface générique (**Storage**) qui définit les méthodes que toutes les classes de stockage doivent respecter :

```
public interface Storage<K, V> {  
    void add(K key, V value);  
    V getValue(K key);  
}
```

Classes d'implémentation

Nous pouvons ensuite créer des classes d'implémentation pour différents types de stockages, par exemple **HashMapStorage** ...

```
public class HashMapStorage<K, V> implements Storage<K, V> {  
    private Map<K, V> storage = new HashMap<>();  
  
    @Override  
    public void add(K key, V value) {  
        storage.put(key, value);  
    }  
  
    @Override  
    public V getValue(K key) {  
        return storage.get(key);  
    }  
}
```

Classes d'implémentation

... ou encore **LinkedListStorage** qui implémentent l'interface générique **Storage** :

```
public class ArrayListStorage<K, V> implements Storage<K, V> {  
    private List<V> storage = new ArrayList<>();  
  
    @Override  
    public void add(K key, V value) {  
        storage.add(new Pair<>(key, value));  
    }  
  
    @Override  
    public V getValue(K key) {  
        for (Pair<K, V> pair : storage) {  
            if (pair.getKey().equals(key)) {  
                return pair.getValue();  
            }  
        }  
        return null;  
    }  
}
```

La classe de gestion

La classe de gestion **DataManager** utilise l'interface générique **Storage** pour stocker et afficher des informations :

```
public class DataManager<K, V> {  
    private Storage<K, V> storage;  
  
    public DataManager(Storage<K, V> storage) {  
        this.storage = storage;  
    }  
  
    public void add(K key, V value) {  
        storage.add(key, value);  
    }  
  
    public V getValue(K key) {  
        return storage.getValue(key);  
    }  
}
```

Avantage

DataManager peut alors être utilisée avec différentes implémentations de stockage (**HashMapStorage** ou **ArrayListStorage**) sans avoir à modifier le code :

```
public class Main {  
    public static void main(String[] args) {  
        // Utilisation de HashMapStorage  
        Storage<String, String> storage = new HashMapStorage<>();  
        DataManager<String, String> dataManager = new DataManager<>(storage);  
        dataManager.add("Jean Saigne", "6 Bd Maréchal Juin");  
  
        // Utilisation d'ArrayListStorage  
        storage = new ArrayListStorage<>();  
        dataManager = new DataManager<>(storage);  
        dataManager.add("Jacques Cepte", "6 Bd Maréchal Juin");  
    }  
}
```

Avantage

La bibliothèque de gestion de données peut être modifiée **sans nécessiter une recompilation** des classes utilisatrices

Il devient facile d'ajouter de nouvelles classes d'implémentation du stockage sans avoir à modifier le code existant, car les interfaces génériques nous permettent de définir des contrats qui doivent être respectés par toutes les classes implémentées

Comment utiliser les lambdas ?

Au fait qu'est-ce que les lambdas en Java ?

```
public class Main {  
    public static void main(String args[]) {
```

Lambdas

```
        Operation addition = (int x, int y) -> x + y; // avec la déclaration de type  
        Operation soustraction = (x, y) -> x - y;      // sans déclaration de type  
        Operation multiplication = (int x, int y) -> { return x * y; }; // 'return' + accolades  
        Operation division = (int x, int y) -> x / y; // sans 'return' et sans les accolades
```

```
        System.out.println("8 + 2 = " + calculer(8, 2, addition));  
        System.out.println("8 - 2 = " + calculer(8, 2, soustraction));  
        System.out.println("8 x 2 = " + calculer(8, 2, multiplication));  
        System.out.println("8 / 2 = " + calculer(8, 2, division));
```

```
    }
```

```
interface Operation {  
    int calc(int x, int y);  
}
```

```
private static int calculer(int x, int y, Operation op) {  
    return op.calc(x, y);  
}
```

```
}
```

Comment mettre en oeuvre les tests avec JUnit ?

Comment tester les exceptions avec JUnit ?

Tester les exceptions sous JUnit 5

```
@Test
public void whenExceptionThrown_thenAssertionSucceeds() {
    try {
        lib.addDocument(d1);
        lib.addDocument(d2);
    } catch (LibraryException e) {
        throw new RuntimeException(e);
    }
```

On suppose une bibliothèque d'une capacité de 2 documents

```
Executable executable = new Executable() {
    @Override
    public void execute() throws Throwable {
        lib.addDocument(d3);
    }
};
```

```
Exception exception = assertThrows(LibraryException.class, executable);
String expectedMessage = "Library has reached her maximal capacity!";
String actualMessage = exception.getMessage();
```

```
assertTrue(actualMessage.contains(expectedMessage));
```

```
}
```

Tester les exceptions sous JUnit 5 (version lambda)

@Test

```
public void whenExceptionThrown_thenAssertionSucceeds() {  
    try {  
        lib.addDocument(d1);  
        lib.addDocument(d2);  
    } catch (LibraryException e) {  
        throw new RuntimeException(e);  
    }  
}
```

```
Executable e = () -> lib.addDocument(d3);
```

```
Exception exception = assertThrows(LibraryException.class, executable);  
String expectedMessage = "Library has reached her maximal capacity!";  
String actualMessage = exception.getMessage();
```

```
assertTrue(actualMessage.contains(expectedMessage));
```

```
}
```

Comment implémenter et synchroniser les threads ?

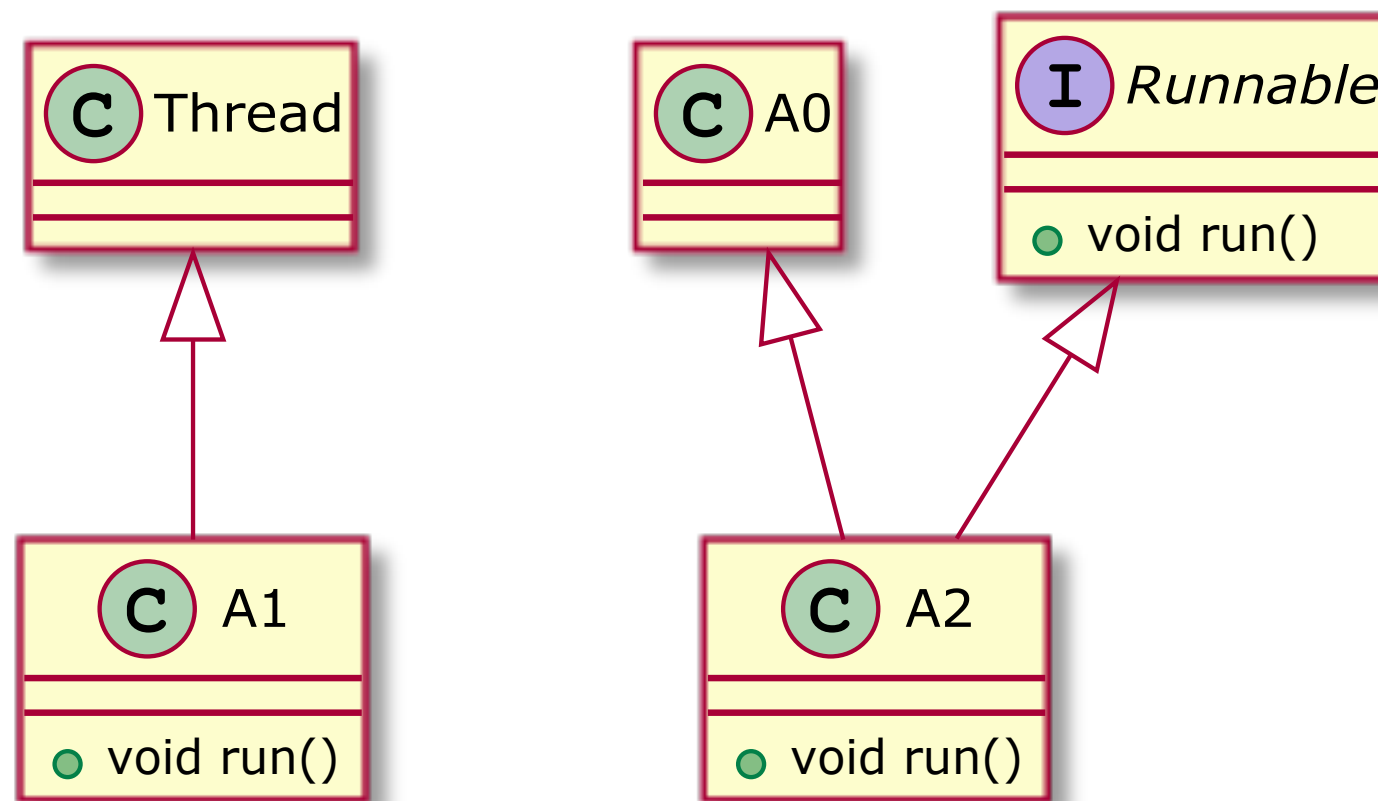
Threads et états

Thread : processus « léger » exécuté au sein d'un processus

- **En cours** (ou **élu**) : invoque la méthode **run()**
- **En attente** : le *thread* est en attente d'exécution
- **Bloqué** : l'ordonnanceur place un *thread* en sommeil (méthode **Thread.sleep()** par exemple)

La commande « *htop* » permet de voir la répartition des threads sur les coeurs du processeur

Création des threads : 2 techniques



Création des threads : 2 techniques

```
public class A1 extends Thread {  
    public A1() {  
    }  
  
    public void run() {  
    }  
}
```

```
public class A2 extends A0 implements Runnable {  
    public A2() {  
    }  
  
    public void run() {  
    }  
}
```

Création des threads : étendre Thread

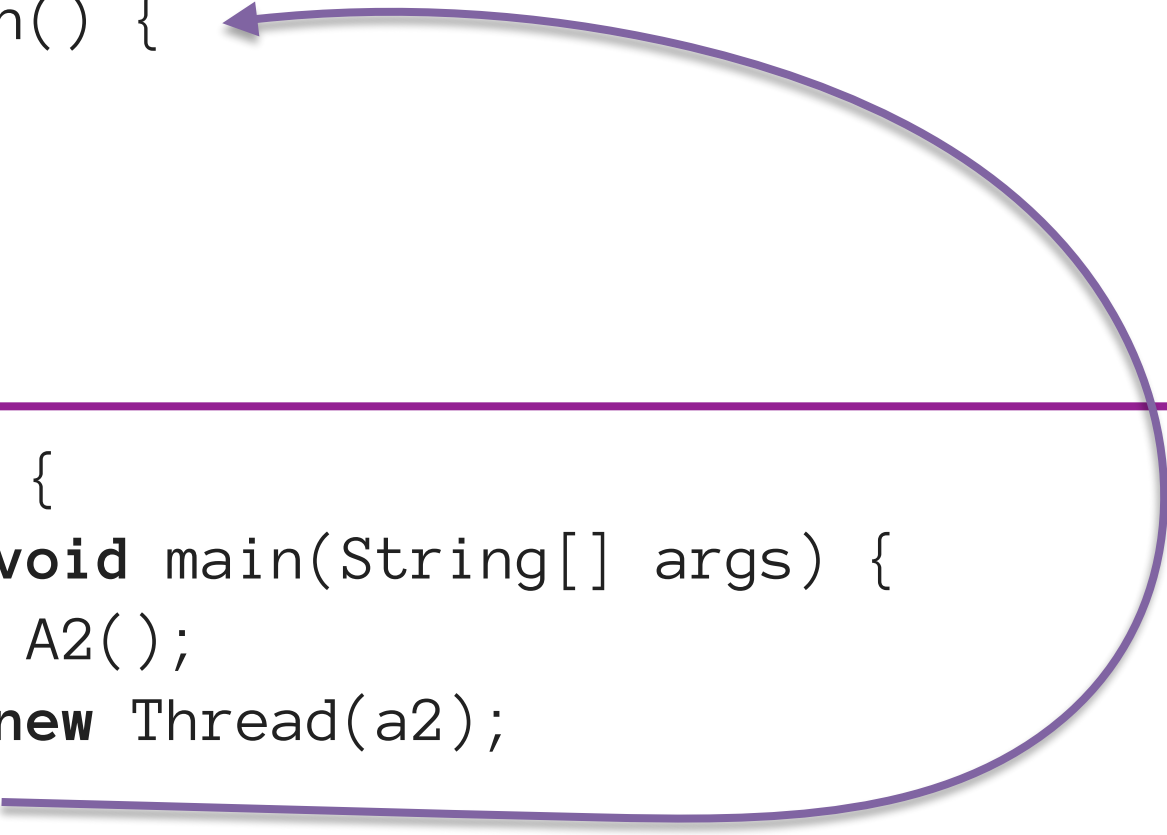
```
public class A1 extends Thread {  
    public A1() {  
    }  
  
    public void run() {  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        A1 a1 = new A1();  
        a1.start();  
    }  
}
```

Création des threads : implémenter Runnable

```
public class A2 extends A0 implements Runnable {  
    public A2() {  
    }  
  
    public void run() {  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        A2 a2 = new A2();  
        Thread t = new Thread(a2);  
        t.start();  
    }  
}
```



Méthodes des threads

`public void run()` : définit l'action réalisée par un *thread*

`public void start()` : lance l'exécution du *thread* (appelle `run()`)

`public void sleep(long milliseconds)` : le *thread* est bloqué pendant quelques ms.

`public void join()` : attente de la fin d'un *thread*

`public void join(long milliseconds)` : attente ciblée de la fin d'un *thread*

`public int getPriority()` : retourne la priorité du *thread*

`public int setPriority(int priority)` : change la priorité du *thread*

`public String getName()` : retourne le nom du *thread*

`public void setName(String name)` : change le nom du *thread*

`public Thread currentThread()` : retourne la référence du *thread* "en cours".

`public int getId()` : retourne l'ID du *thread*

`public Thread.State getState()` : retourne l'état du *thread*

`public boolean isAlive()` : teste si le *thread* est en vie

`public void yield()` : bloque le *thread* "en cours".

`public boolean isDaemon()` : teste si le *thread* est un démon

`public void setDaemon(boolean b)` : transforme le *thread* en démon

`public void interrupt()` : bloque le *thread*

`public boolean isInterrupted()` : teste si le *thread* a été bloqué

`public static boolean interrupted()` : teste si le *thread* "en cours" a été bloqué

Threads et synchronisation

L'accès à une « ressource » partagée par plusieurs *threads* nécessite une **synchronisation**

```
public class Ressource {
    private int result;
    private boolean computed = false;

    public void sum(int n) {
        int sum = 0;
        for (int i = 0; i < 10000; i++) {
            sum = n + i;
        }
        result = sum;
        computed = true;
    }

    public int getResult() {
        while (!computed) {
            try {

            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
        computed = false;
        return result;
    }
}
```

Threads et synchronisation

```
public class A1 extends Thread {  
    Ressource r;  
  
    public A1(Ressource r) {  
        this.r = r;  
    }  
  
    public void run() {  
        r.sum(10);  
        System.out.println("" + r.getResult());  
    }  
}
```

```
public class A2 extends Thread {  
    Ressource r;  
  
    public A2(Ressource r) {  
        this.r = r;  
    }  
  
    public void run() {  
        r.sum(200);  
        System.out.println("" + r.getResult());  
    }  
}
```

Threads et synchronisation

L'accès à une « ressource » partagée par plusieurs *threads* nécessite une **synchronisation**

```
public class Ressource {
    private int result;
    private boolean computed = false;

    public synchronized void sum(int n) {
        int sum = 0;
        for (int i = 0; i < 10000; i++) {
            sum = n + i;
        }
        result = sum;
        computed = true;
        notifyAll();
    }

    public synchronized int getResult() {
        while (!computed) {
            try {
                wait();
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
        computed = false;
        return result;
    }
}
```


Comment manipuler les expressions régulières ?

Expressions régulières : exemples

- Chaîne spécifique : **bon**
- Chaîne avec des caractères spécifiques : ***[@/:]***
- Chaîne avec alphabet uniquement : ***[a-z]***
- Chaîne sans chiffres : ***[^0-9]***

<https://www.regexpal.com/>

Expressions régulières : exemples

<code>.</code>	tout caractère excepté passage à la ligne
<code>\w \d \s</code>	mot, chiffre, espace
<code>\W \D \S</code>	pas un mot, chiffre, espace
<code>[abc]</code>	a, b, ou c
<code>[^abc]</code>	ni a, ni b, ni c
<code>[a-g]</code>	caractères entre a et g
<code>^abc\$</code>	début / fin de chaîne

Quantificateurs

<code>a* a+ a?</code>	0 ou plus, 1 ou plus, 0 ou 1
<code>a{5} a{2,}</code>	exactement 5, 2 ou plus
<code>a{1,3}</code>	entre 1 et 3
<code>ab cd</code>	cherche ab ou cd

Caractères d'échappement

<code>\. * \\</code>	échappement des caractères spéciaux
<code>\t \n \r</code>	tabulation, passage à la ligne, retour chariot
<code>\u00A9</code>	caractère unicode ©

Expressions régulières : exemple avec String.split()

```
String in = "azerty/2023/jsaigne@ensicaen.fr:80";
```

```
String[] parts = in.split("[/@:]");
```

```
// "azerty", "2023", "jsaigne", "ensicaen.fr", "80"
```

```
String out = in.replaceAll("[@/:] ", " ");
```

```
// "azerty 2023 jsaigne ensicaen.fr 80"
```

```
String charsToRemove = "/@:";
```

```
for (int i = 0; i < charsToRemove.length(); i++) {
```

```
    str = in.replace(charsToRemove.charAt(i) + "", "");
```

```
}
```

```
// "azerty2023jsaigneensicaen.fr80"
```

Expressions régulières : exemples avec Regex/Matcher


```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

String in = "azerty/2023/jsaigne@ensicaen.fr:80";
Pattern pattern = Pattern.compile("[/@:]",
                                   Pattern.CASE_INSENSITIVE);
Matcher matcher = pattern.matcher(in);
boolean matchFound = matcher.find();
if (matchFound) {
    System.out.println("Motif trouvé");
} else {
    System.out.println("Motif non trouvé");
}
```

Expressions régulières : exemples avec Regex/Matcher

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

String in = "azerty/2023/jsaigne@ensicaen.fr:80";
Pattern pattern = Pattern.compile("[/@:]",
                                   Pattern.LITERAL);
Matcher matcher = pattern.matcher(in);
boolean matchFound = matcher.find();
if (matchFound) {
    System.out.println("Motif trouvé");
} else {
    System.out.println("Motif non trouvé");
}
```



Caractères d'échappement non pris en compte !

Expressions régulières : un courriel

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

String in = "jsaigne@ensicaen.fr";
String pattern =
    "^([a-zA-Z0-9_\\-\\.]+)@([a-zA-Z0-9_\\-\\.]+)\\.([a-zA-Z]{2,5})$"

Pattern pattern = Pattern.compile(pattern, Pattern.CASE_INSENSITIVE);
Matcher matcher = pattern.matcher(in);
boolean matchFound = matcher.find();

if (matchFound) {
    System.out.println("C'est un courriel");
} else {
    System.out.println("Le format est incorrect");
}
```