III. Conception de classes en Java

III.6 Gestion des erreurs, exceptions

Gestion des erreurs - exceptions

- Erreurs dues au programmeur …
  - Programmation incorrecte
    - Non conformité à la spécification
  - Utilisation inappropriée d'un objet
    - P.ex. index de tableau ou de liste invalide
  - État d'objet incohérent
- … mais pas seulement
  - Erreurs dues à l'environnement
    - URL invalide
    - Coupure réseau
  - Cas particulier : traitement de fichiers
    - Fichiers absents
    - Droits d'accès inappropriés

Programmation “défensive”

- Un objet sera réalisé différement selon qu’on considère
  - qu’il sera toujours utilisé d’une manière “correcte”
  - ou bien que des mauvaises utilisations sont possibles
- Dans une interaction de type “client-serveur” (appelant-appelé): un objet “client” utilise un objet “ serveur” *via* ses méthodes
  - Jusqu’à quel point les méthodes du serveur doivent-elles effectuer des vérifications?
  - Comment signaler les erreurs au client?
  - Comment un client peut anticiper l’échec d’une requête au serveur?
  - Comment doit réagir le client en cas d’échec d’une requête?

Exemple : la classe CarnetAdresses

```
import java.util.ArrayList;


public class CarnetAdresses {
  private ArrayList<Entrée> carnet;

  public CarnetAdresses() {
   carnet = new ArrayList<Entrée>();
  }

  public void addEntrée(String nom, String   adresse, String téléphone) {
   Entrée e = new Entrée(nom, adresse, téléphone);

   if (nom.equals("")){
     // Erreur
   }

   else if (nomExiste(nom)){
     // Erreur
   }

   else carnet.add(e);
  }
}
```

Exemple : la classe CarnetAdresses

```
public static void main(String args[]){
 CarnetAdresses c = new CarnetAdresses();

  c.addEntrée("ike", "ici", "123");
  c.addEntrée("", "ici", "255"); // Erreur
  c.addEntrée("ike", "ici", "255"); // Erreur

}
```

Exemple

```
c.addEntrée("ike", "ici", "123");
c.addEntrée("", "ici", "255"); // Erreur
c.addEntrée("ike", "ici", "255"); // Erreur
```

- Une erreur se produit à l'exécution
  - A qui la faute?
  - Comment la prévenir?

Valeurs des paramètres

- Les paramètres représentent un facteur majeur de ‘vulnerabilité’ pour un objet utilisé.
  - Les paramètres du constructeur servent à initialiser l’état de l’objet
  - Les paramètres des méthodes peuvent modifier cet état
- La vérification de la valeur des paramètres est une mesure défensive.

Exemple : la classe CarnetAdresses

```
import java.util.ArrayList;


public class CarnetAdresses {
  private ArrayList<Entrée> carnet;

  public CarnetAdresses() {
   carnet = new ArrayList<Entrée>();
  }

  public void addEntrée(String nom, String   adresse, String téléphone) {
   Entrée e = new Entrée(nom, adresse, téléphone);

   if (nom.equals("")){
     // Erreur
   }

   else if (nomExiste(nom)){
     // Erreur
   }

   else carnet.add(e);
  }
}
```

Signaler les erreurs

- Comment signaler la présence d'arguments inappropriés et à qui?
  - A l'utilisateur?
    - En affichant un message d'erreur
    - Mais y a-t-il un utilisateur humain?
    - Peut-il vraiment résoudre le problème?

Exemple : la classe CarnetAdresses

```
import java.util.ArrayList;


public class CarnetAdresses {
  private ArrayList<Entrée> carnet;

  public CarnetAdresses() {
   carnet = new ArrayList<Entrée>();
  }

  public void addEntrée(String nom, String   adresse, String téléphone) {
   Entrée e = new Entrée(nom, adresse, téléphone);

   if (nom.equals("")){
     System.out.println("nom invalide");
   }

   else if (nomExiste(nom)){
     System.out.println("nom existant");    }

   else carnet.add(e);
  }
}
```

Signaler les erreurs

- Comment signaler la présence d'arguments inappropriés et à qui?
  - A l'objet client?
    - Retourner un code d'erreur?

Exemple : la classe CarnetAdresses

```
import java.util.ArrayList;


public class CarnetAdresses {
  private ArrayList<Entrée> carnet;

  public CarnetAdresses() {
   carnet = new ArrayList<Entrée>();
  }

  public int addEntrée(String nom, String   adresse, String téléphone) {
   Entrée e = new Entrée(nom, adresse, téléphone);

   if (nom.equals("")){
     return 1;
   }

   else if (nomExiste(nom)){
     return 2;    }

   else {
     carnet.add(e);
     return 0;    }
   }
  }
}
```

Que peut faire le client (méthode appelante)?

- Tester la valeur retournée
  - Tenter de réparer l'erreur et d'éviter une défaillance du programme.

```
if (c.addEntrée("ike", "ici", "123") == 1) {…} else {…};
if (c.addEntrée("", "ici", "255") == 1) {…} else {…}; // Erreur
if (c.addEntrée("ike", "ici", "255") == 1) {…} else {…}; // Erreur
```

- Ignorer la valeur retournée

```
c.addEntrée("ike", "ici", "123");
c.addEntrée("", "ici", "255"); // Erreur
c.addEntrée("ike", "ici", "255"); // Erreur
```

  - Une défaillance est possible.

- L'objet appelé n'a aucun moyen d'obliger l'appelant de tester la valeur retournée…
- Autre solution (préférable) : ***déclencher une exception***

Principes du déclenchement des exceptions

- Exceptions: une caractéristique spécifique du langage.
- Ne nécessitent pas de valeur de retour “spéciale”
- Les erreurs **ne peuvent pas être ignorées** par le client.
  - L’exception interrompt le flux de contrôle du programme
- Mécanisme encourageant le client de définir des actions spécifiques de traitement des exceptions.

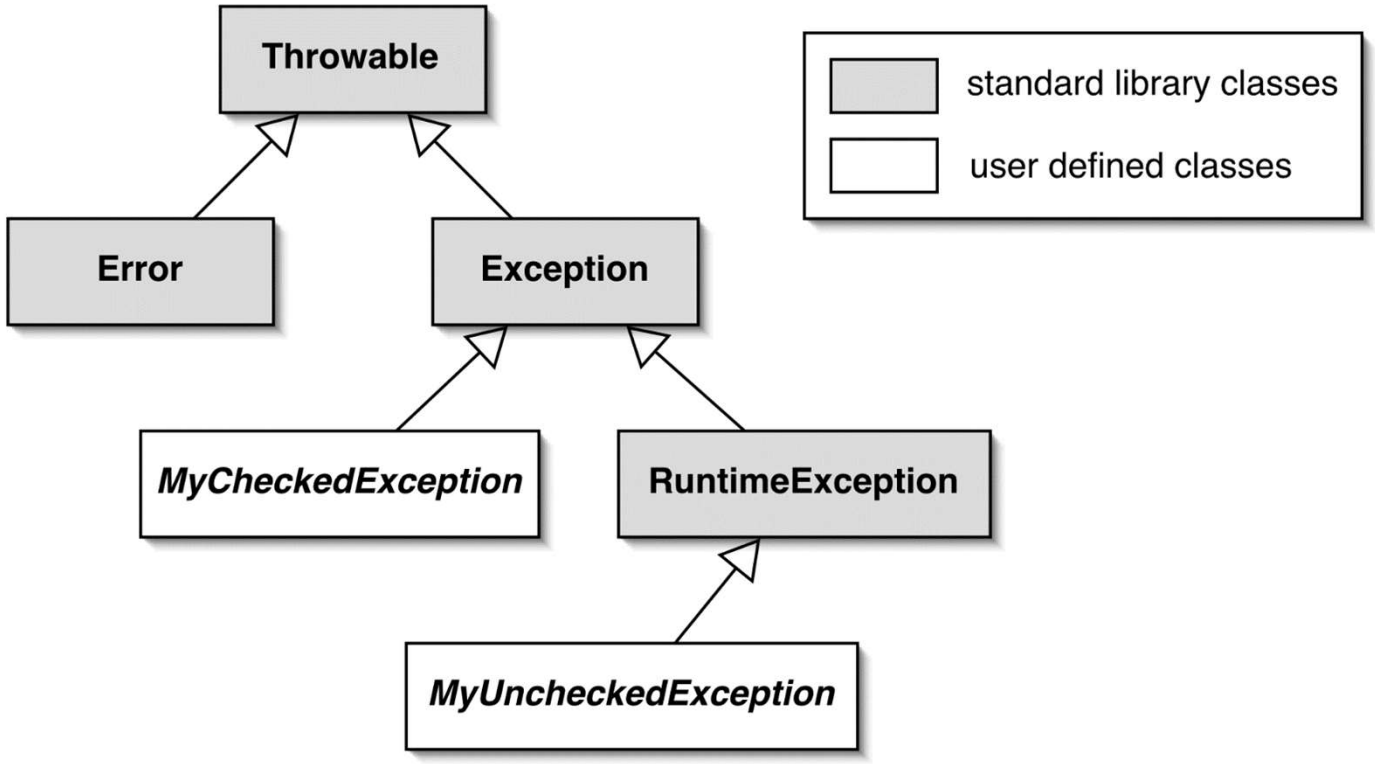
Déclenchement d'une exception

```
/**
 * @param nom Nom du contact.
 * @param adrsese Adresse du contact.
 * @param Téléphone Numéro du contact.
 * @throws Exception si insertion échoue.
 */

public void addEntrée(String nom, String adresse, String téléphone)
                                        throws Exception {
  Entrée e = new Entrée(nom, adresse, téléphone);

  if (nom.equals("") || nomExiste(nom)){
    throw new Exception (nom);
  }

  carnet.add(e);
}
```

Déclenchement d'une exception

- An objet exception est construit
  - `new ExceptionType("...");`
- L'exception est déclenchée
  - `throw ...`
- Commentaire Javadoc associé:
  - `@throws ExceptionType reason`

Hiérarchie des classes d'exception



Catégories d'exceptions

- Exceptions sous contrôle
  - Sous classe de `Exception`
  - Utilisées pour des erreurs qu'on peut anticiper…
  - … et où une action de récupération est possible
- Exceptions hors contrôle
  - Sous classe de `RuntimeException`
  - Utilisées pour des erreurs qu'on ne peut pas prévoir…
  - … et où une action de récupération n'est pas possible

Effet du déclenchement d'une exception

- La méthode M ayant déclenché l'exception
  - se termine immédiatement
  - ne retourne aucune valeur

- L'exécution ne reprend pas là où le client a appelé la méthode M
  - Le client ne peut pas poursuivre son exécution comme si rien n'était.

- Le client peut ***capturer*** l'exception

Exceptions hors contrôle

- Exceptions dont l'emploi n'est pas vérifié par le compilateur
- Elles provoquent la terminaison du programme si elles ne sont pas capturées
  - Ce n'est pas une bonne pratique de programmation
- Exemple : `IllegalArgumentException`

Exemple

```
import java.util.ArrayList;


public class CarnetAdresses {
  private ArrayList<Entrée> carnet;

  public CarnetAdresses() {
   carnet = new ArrayList<Entrée>();
  }

  public void addEntrée(String nom, String   adresse, String téléphone) {
   Entrée e = ...;


   if (e == null){
     throw new NullPointerException(nom);
   }

   // ...

   carnet.add(e);

  }
}
```

Gestion des exception sous contrôle

- Les exceptions sous contrôle sont sensées être capturées.
- Le compilateur vérifie leur utilisation.
  - Côte client ET côté serveur.
- Les erreurs à l'origine des exceptions peuvent être réparées.

La clause `throws`

Déclaration du fait qu'une méthode peut déclencher une exception

```
public void addEntrée(String nom, String adresse,
                      String téléphone) throws Exception
```

L'instruction try

- L'objet client doit protéger l'appel à une méthode du serveur susceptible de déclencher une exception:

```
try {
    // Instructions protégées
}
catch(Exception e) {
    // Traitement de l'exception
}
```

1. Exception déclenchée ici

```
try {
  c.addEntrée("ike", "ici", "123");
  c.addEntrée("", "ici", "255");
  c.addEntrée("ike", "ici", "255");
 }
 catch(Exception e){
  System.out.println ("invalide " + e);
 }
```

2. Reprise de l'exécution ici

Definition d'exceptions

```
public class ExceptionElementExistant extends Exception
{
    private String key;

    public ExceptionElementExistant(String key)
    {
        this.key = key;
    }

    public String getKey()
    {
        return key;
    }

    public String toString()
    {
        return "L'élément " + key + " est déjà dans la liste";

    }
}
```

Exceptions multiples

```
import java.util.ArrayList;


public class CarnetAdresses {
  private ArrayList<Entrée> carnet;

  public CarnetAdresses() {
   carnet = new ArrayList<Entrée>();
  }

  public void addEntrée(String nom, String   adresse, String téléphone) throws
                                                ExceptionElementInvalide, ExceptionElementExistant{
   Entrée e = new Entrée(nom, adresse, téléphone);

   if (nom.equals("")){
     throw new ExceptionElementInvalide(nom);
   }

   if (nomExiste(nom)){
     throw new ExceptionElementExistant(nom);
   }

   carnet.add(e);

  }
}
```

Exceptions multiples

```
try {
   c.addEntrée("ike", "ici", "123");
   c.addEntrée("", "ici", "255");
   c.addEntrée("ike", "ici", "255");

 }
 catch(ExceptionElementInvalide ei){
  System.out.println ("invalide " + ei);
 }
 catch(ExceptionElementExistant ee){
  System.out.println ("existant " + ee);
 }
```

La clause `finally`

```
try {
    // Instructions protégées
}
catch(Exception e) {
    // Traitement de l'exception
}
finally {
    // Instruction à exécuter
    // qu'il y ait eu exception ou pas
}
```

- La clause finally s'exécute même si une instruction `return` est exécutée dans les clauses `try` ou `catch`
- La clause finally s'exécute même en cas d'exception non capturée

La clause `finally`

(exemple, source : The Java™ Tutorials)

```
static String readFirstLineFromFileWithFinallyBlock(String path)
                                                    throws IOException {
    BufferedReader br = new BufferedReader(new FileReader(path));
    try {
        return br.readLine();
    }
    catch(IOException e) {
       // Traitement de l'exception
    }
    finally {
        if (br != null) br.close();
    }
}
```

- Finally permet d'assurer la libération "propre" d'éventuelles ressources utilisées, malgré une possible interruption.

Exemple

```
public static void main(String args[]){
 CarnetAdresses c = new CarnetAdresses();

 try {
  c.addEntrée("ike", "ici", "123");
  c.addEntrée("", "ici", "255");
  c.addEntrée("ike", "ici", "255");

 }
 catch(ExceptionElementInvalide ei){
  System.out.println ("invalide " + ei);
 }
 catch(ExceptionElementExistant ee){
  System.out.println ("existant " + ee);
 }
 finally{
  System.out.println ("Finally executé");
 }
}
```

Les assertions

- Utilisées pour des contrôles internes de cohérence
  - p.ex. vérifier la cohérence de l'état de l'objet
  - Les assertions ne sont pas là pour remplacer les exceptions.
- Utilisées lors du développement mais supprimées dans la version finale du programme
- Java possède une instruction `assert`
  - Deux utilisations sont possibles
    - **`assert`** expression-*boolénne*
    - **`assert`** expression-*boolénne* **:** *expression*
  - L'expression booléenne définit une condition qui doit être vraie au moment de l'évaluation de l'assertion
  - Une exception de type `AssertionError` est déclénchée si l'assertion est fausse

Assert

```
public void removeDetails(String key)
    {
        if(key == null){
            throw new IllegalArgumentException("...");
        }
        if(keyInUse(key)) {
            ContactDetails details = book.get(key);
            book.remove(details.getName());
            book.remove(details.getPhone());
            numberOfEntries--;
        }
        assert !keyInUse(key);
        assert consistentSize() :
                "Inconsistent book size in removeDetails";
    }
```

Bilan des concepts introduits

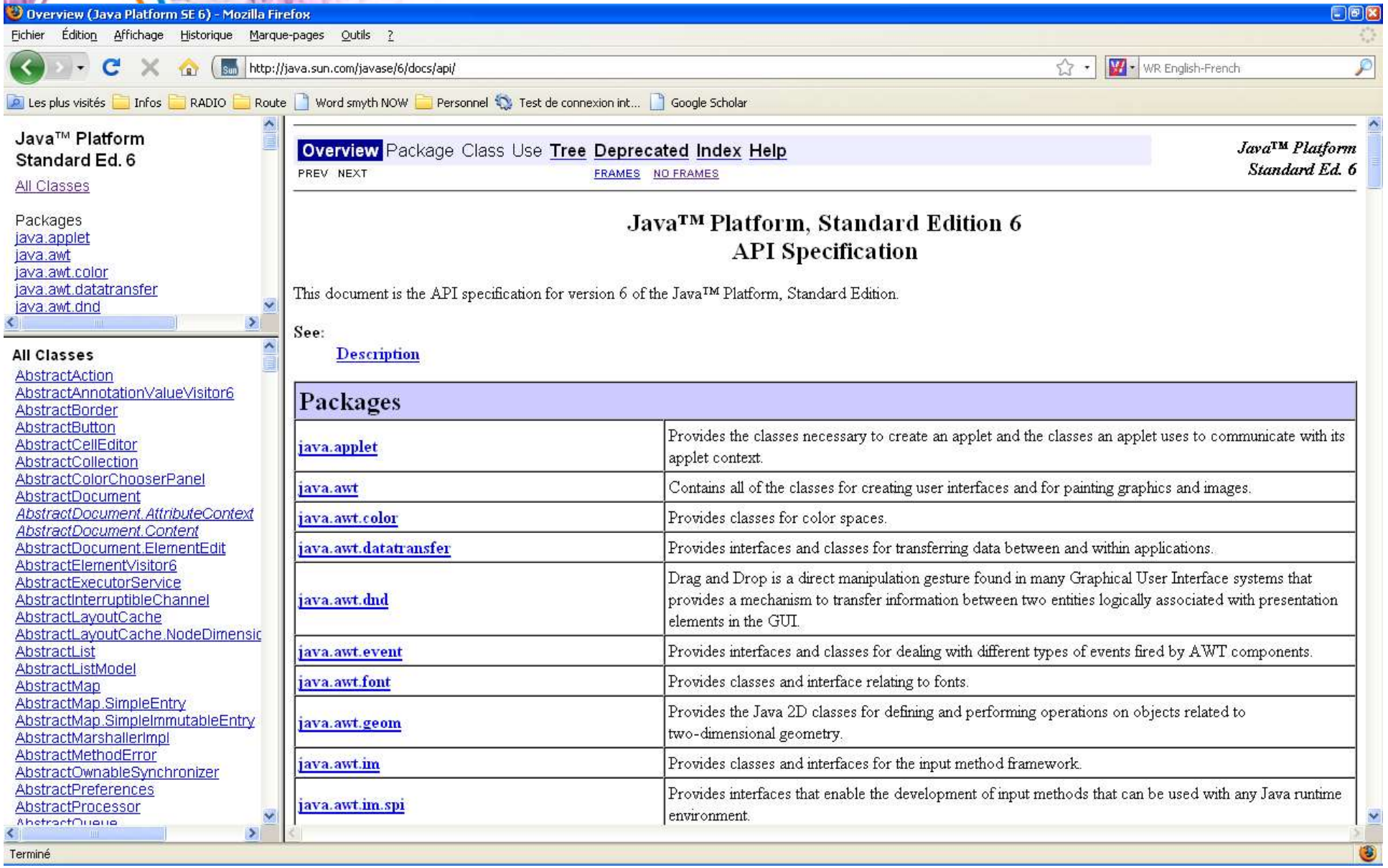
- Erreurs et programmation défensive
- Exceptions : principe
- throw et throws
- try - catch - finally

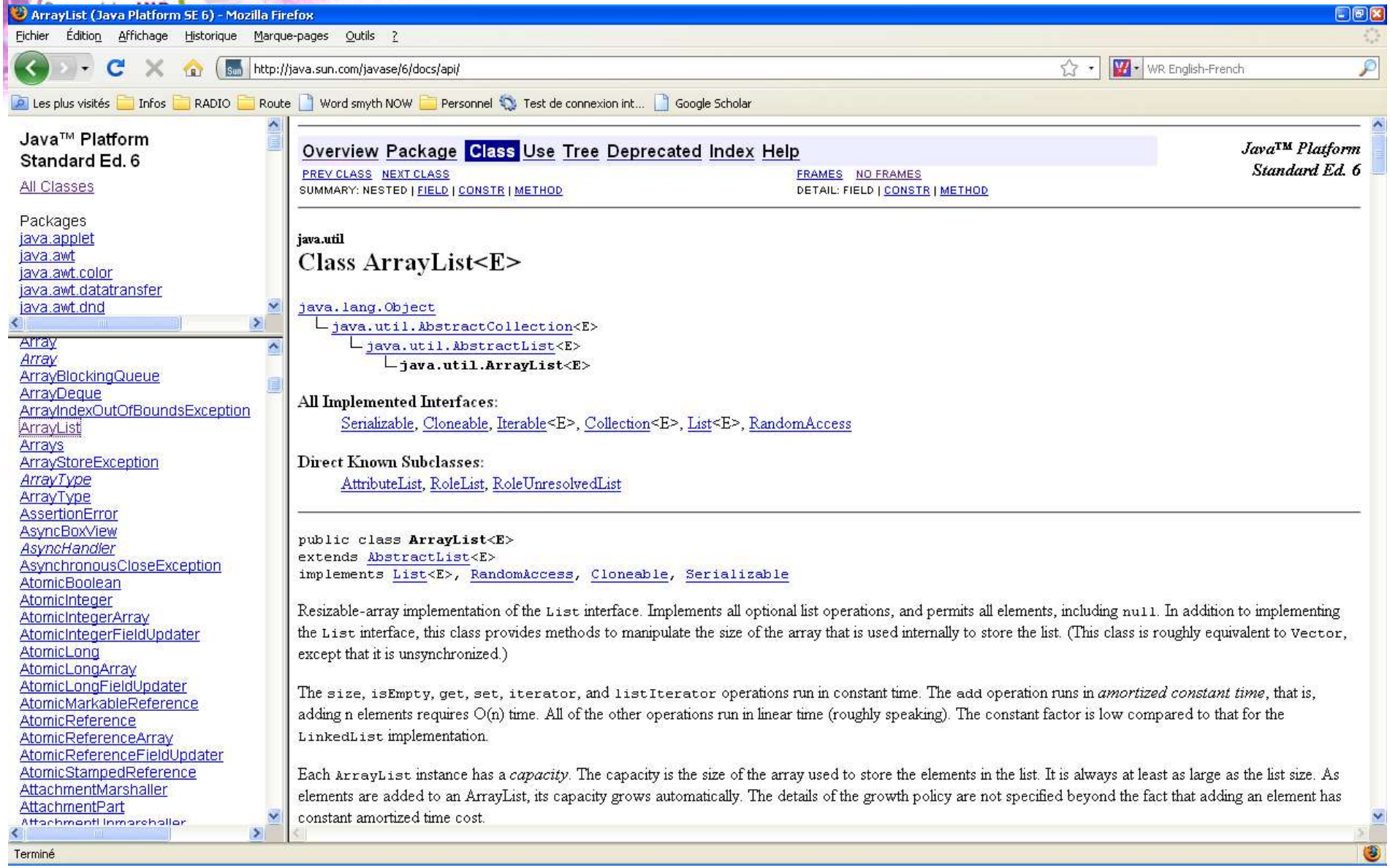
III. Conception de classes en Java

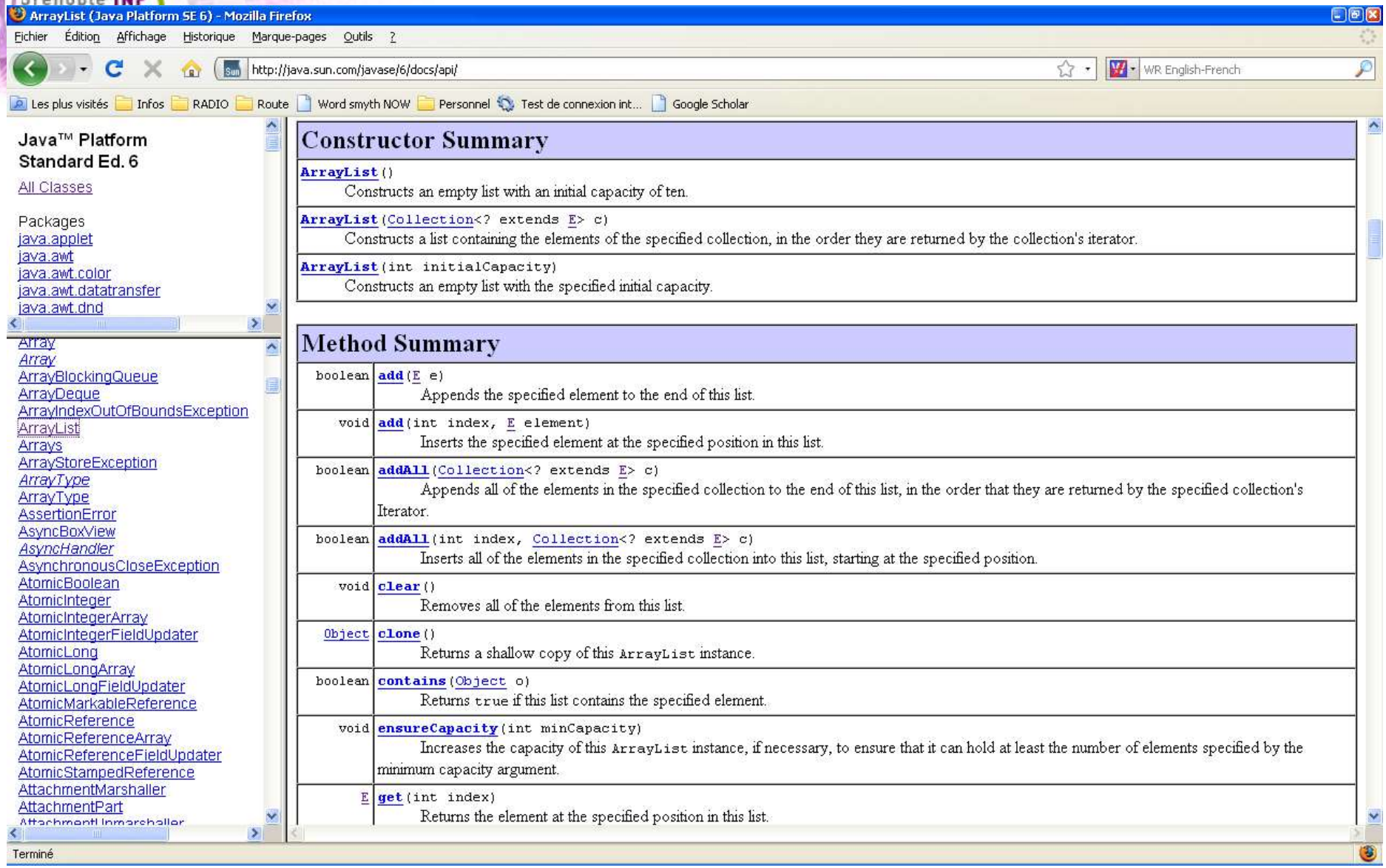
III.6 Librairie Java

La librairie Java

- Des milliers de classes utiles qui rendent la programmation beaucoup plus facile
- Un bon développeur Java DOIT savoir utiliser des librairies
  - Connaître par coeur certaines classes (leur nom)
  - Savoir comment chercher des informations sur d'autres classes
    - Seule l'interface (méthodes publiques) de la classe doit être connue pour son utilisation (l'implémentation est sans intérêt)
- Documentation en format HTML
  - Accessible avec un navigateur web
  - Description de l'interface de toutes les classes


Overview (Java Platform SE 6) - Mozilla Firefox
Fichier Édition Affichage Historique Marque-pages Outils ?
http://java.sun.com/javase/6/docs/api/
WR English-French
Les plus visités Infos RADIO Route Word smyth NOW Personnel Test de connexion int... Google Scholar
Java™ Platform Standard Ed. 6
All Classes
Packages
java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
All Classes
AbstractAction
AbstractAnnotationValueVisitor6
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeContext
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractElementVisitor6
AbstractExecutorService
AbstractInterruptibleChannel
AbstractLayoutCache
AbstractLayoutCache.NodeDimensic
AbstractList
AbstractListModel
AbstractMap
AbstractMap.SimpleEntry
AbstractMap.SimpleImmutableEntry
AbstractMarshallerImpl
AbstractMethodError
AbstractOwnableSynchronizer
AbstractPreferences
AbstractProcessor
Overview Package Class Use Tree Deprecated Index Help
PREV NEXT
FRAMES NO FRAMES
Java™ Platform Standard Ed. 6
Java™ Platform, Standard Edition 6
API Specification
This document is the API specification for version 6 of the Java™ Platform, Standard Edition.
See:
Description
Packages
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 fired by AWT components.
java.awt.font
Provides classes and interface relating to fonts.
java.awt.geom
Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im
Provides classes and interfaces for the input method framework.
java.awt.im.spi
Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
Terminé


ArrayList (Java Platform SE 6) - Mozilla Firefox
Fichier Édition Affichage Historique Marque-pages Outils ?
http://java.sun.com/javase/6/docs/api/
Les plus visités Infos RADIO Route Word smyth NOW Personnel Test de connexion int... Google Scholar
Java™ Platform Standard Ed. 6
All Classes
Packages
java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
Array
Array
ArrayBlockingQueue
ArrayDeque
ArrayIndexOutOfBoundsException
ArrayList
Arrays
ArrayStoreException
ArrayType
ArrayType
AssertionError
AsyncBoxView
AsyncHandler
AsynchronousCloseException
AtomicBoolean
AtomicInteger
AtomicIntegerArray
AtomicIntegerFieldUpdater
AtomicLong
AtomicLongArray
AtomicLongFieldUpdater
AtomicMarkableReference
AtomicReference
AtomicReferenceArray
AtomicReferenceFieldUpdater
AtomicStampedReference
AttachmentMarshaller
AttachmentPart
Overview Package Class Use Tree Deprecated Index Help
PREV CLASS NEXT CLASS
FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD
Java™ Platform Standard Ed. 6
java.util
Class ArrayList<E>
java.lang.Object
java.util.AbstractCollection<E>
java.util.AbstractList<E>
java.util.ArrayList<E>
All Implemented Interfaces:
Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess
Direct Known Subclasses:
AttributeList, RoleList, RoleUnresolvedList
public class ArrayList<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable
Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized.)
The size, isEmpty, get, set, iterator, and listIterator operations run in constant time. The add operation runs in amortized constant time, that is, adding n elements requires O(n) time. All of the other operations run in linear time (roughly speaking). The constant factor is low compared to that for the LinkedList implementation.
Each ArrayList instance has a capacity. The capacity is the size of the array used to store the elements in the list. It is always at least as large as the list size. As elements are added to an ArrayList, its capacity grows automatically. The details of the growth policy are not specified beyond the fact that adding an element has constant amortized time cost.
Terminé

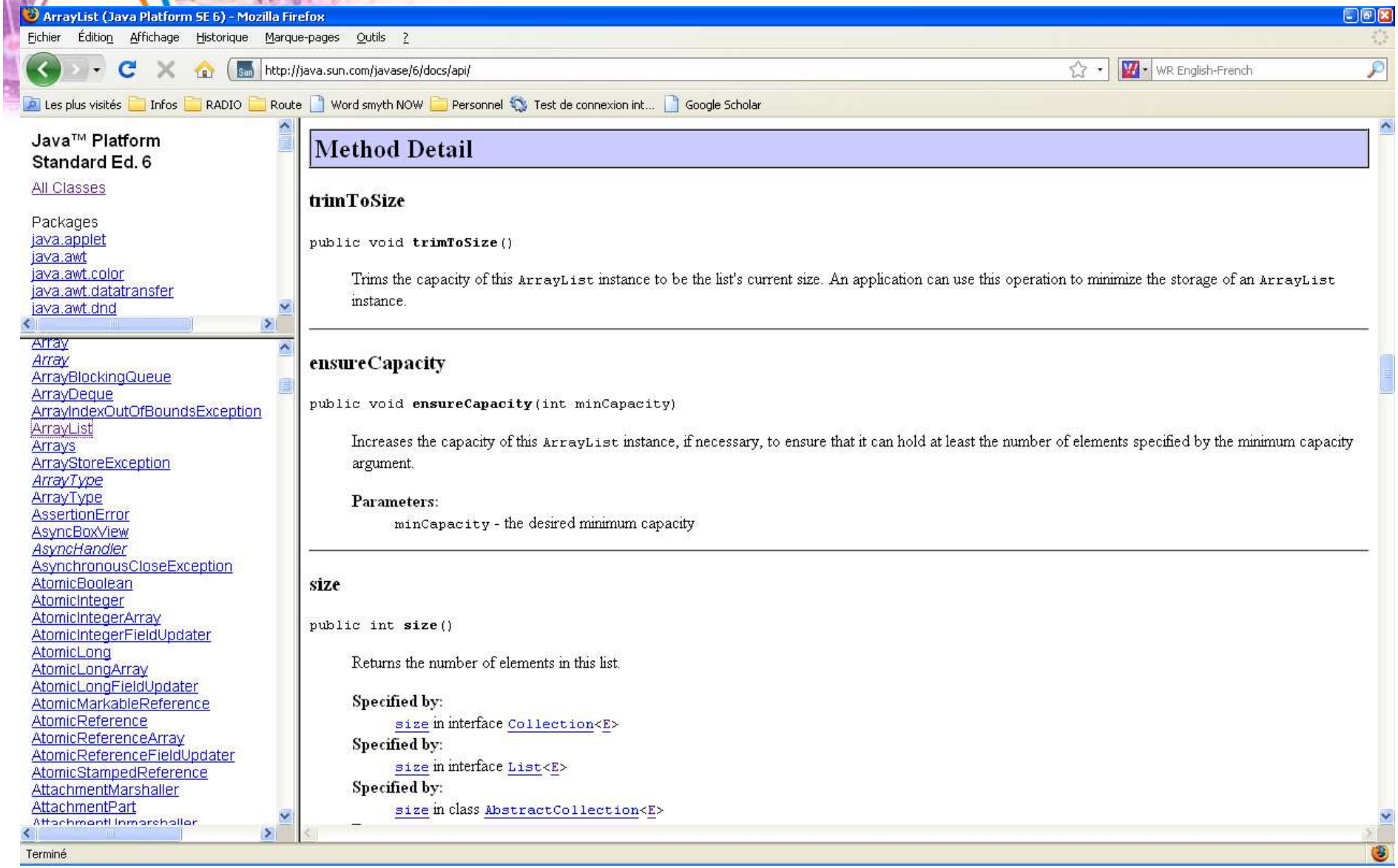


Constructor Summary

ArrayList()	Constructs an empty list with an initial capacity of ten.
ArrayList(Collection<? extends E> c)	Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
ArrayList(int initialCapacity)	Constructs an empty list with the specified initial capacity.

Method Summary

boolean	**add**(E e) Appends the specified element to the end of this list.
void	**add**(int index, E element) Inserts the specified element at the specified position in this list.
boolean	**addAll**(Collection<? extends E> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's Iterator.
boolean	**addAll**(int index, Collection<? extends E> c) Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	**clear**() Removes all of the elements from this list.
Object	**clone**() Returns a shallow copy of this ArrayList instance.
boolean	**contains**(Object o) Returns true if this list contains the specified element.
void	**ensureCapacity**(int minCapacity) Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
E	**get**(int index) Returns the element at the specified position in this list.


ArrayList (Java Platform SE 6) - Mozilla Firefox
Fichier Édition Affichage Historique Marque-pages Outils ?
http://java.sun.com/javase/6/docs/api/
Les plus visités Infos RADIO Route Word smyth NOW Personnel Test de connexion int... Google Scholar
Java™ Platform Standard Ed. 6
All Classes
Packages
java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
Array
Array
ArrayBlockingQueue
ArrayDeque
ArrayIndexOutOfBoundsException
ArrayList
Arrays
ArrayStoreException
ArrayType
ArrayType
AssertionError
AsyncBoxView
AsyncHandler
AsynchronousCloseException
AtomicBoolean
AtomicInteger
AtomicIntegerArray
AtomicIntegerFieldUpdater
AtomicLong
AtomicLongArray
AtomicLongFieldUpdater
AtomicMarkableReference
AtomicReference
AtomicReferenceArray
AtomicReferenceFieldUpdater
AtomicStampedReference
AttachmentMarshaller
AttachmentPart
Method Detail
trimToSize
public void trimToSize()
Trims the capacity of this ArrayList instance to be the list's current size. An application can use this operation to minimize the storage of an ArrayList instance.
ensureCapacity
public void ensureCapacity(int minCapacity)
Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
Parameters:
minCapacity - the desired minimum capacity
size
public int size()
Returns the number of elements in this list.
Specified by:
size in interface Collection<E>
Specified by:
size in interface List<E>
Specified by:
size in class AbstractCollection<E>
Terminé

Interface vs implémentation

La documentation inclut

- La nom de la classe
- Une description informelle de la classe
- La liste des signatures des constructeurs et des méthodes
- Une description informelle du service rendu par la méthode

l'*interface* de la classe

La documentation n'inclut pas

- Les attributs (privés)
- Les méthodes privées
- Le code source de chaque méthode

l'*implémentation* de la classe

Packages et `import`

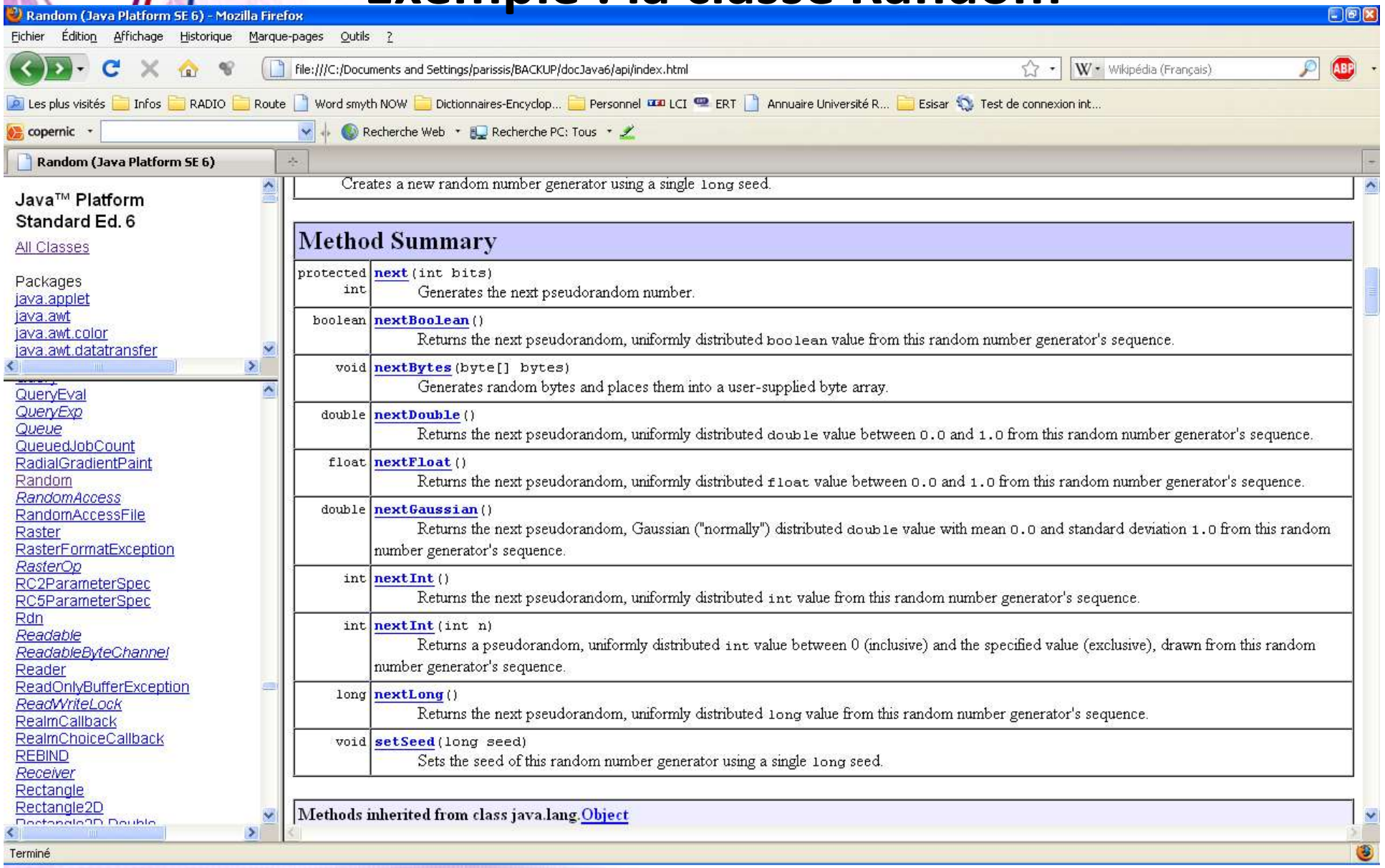
- Avant d'utiliser une classe de la librairie Java, il faut *l'importer*
  - (à l'exception des classes de *java.lang* importées par défaut)
- Une fois importées, elles sont utilisées comme toute autre classe
- Les classes sont organisées en "packages" (paquetages)
  - On peut importer des classes

    ```
    import java.util.ArrayList;
    ```

  - Ou bien des packages entiers (i.e. toutes les classes d'un package)

    ```
    import java.util.*;
    ```

Exemple : la classe Random



Exemple : la classe Random

- La classe `Random` permet de générer des nombres aléatoires

```
import java.util.Random;
...
Random randomGenerator = new Random();
...
int index1 = randomGenerator.nextInt();
int index2 = randomGenerator.nextInt(100);
```

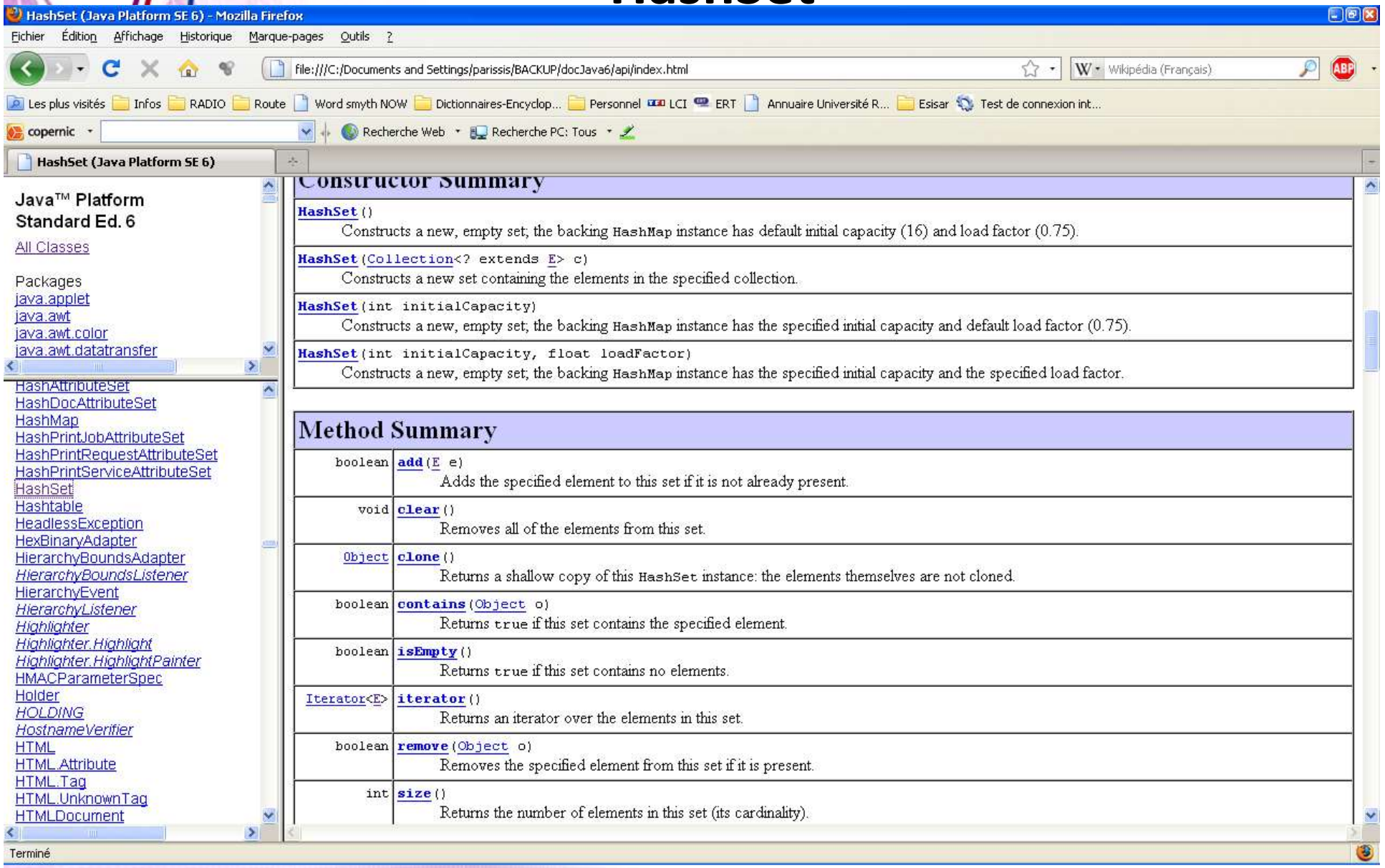
Que fait la méthode generateResponse?

```
public Responder()
{
    randomGenerator = new Random();
    responses = new ArrayList<String>();
    fillResponses();
}

public String generateResponse()
{
    int index = randomGenerator.nextInt(responses.size());
    return responses.get(index);
}

public void fillResponses()
{
    ...
}
```

Classe HashSet : manipulation d'ensembles

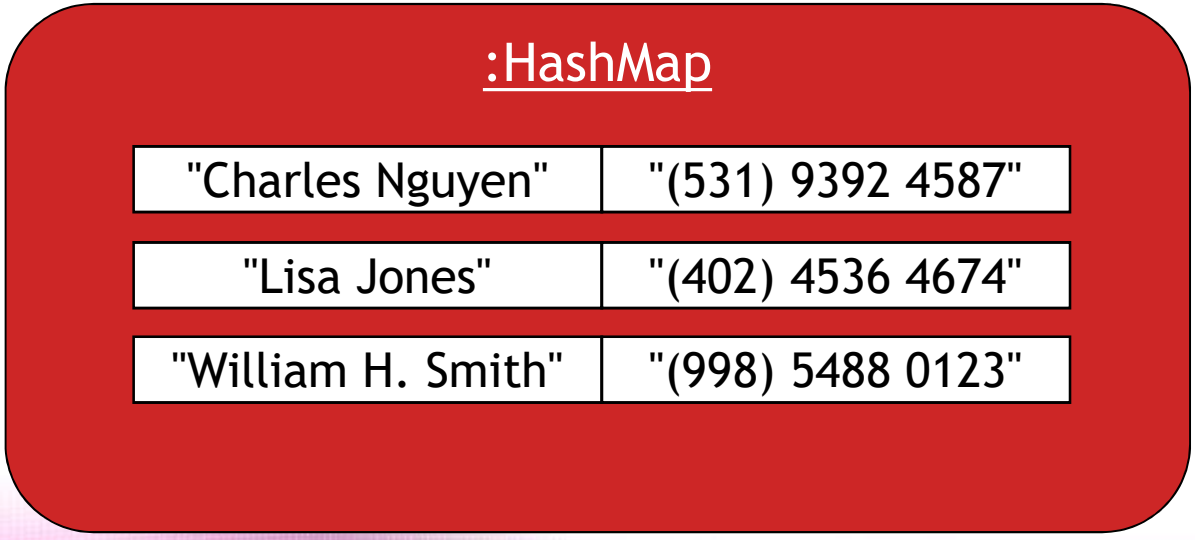
```
import java.util.HashSet;

...
HashSet<String> mySet = new HashSet<String>();
...
mySet.add("un");
mySet.add("deux");
mySet.add("trois");

for(String element : mySet) {
    System.out.println(element);
}
```

HashSet

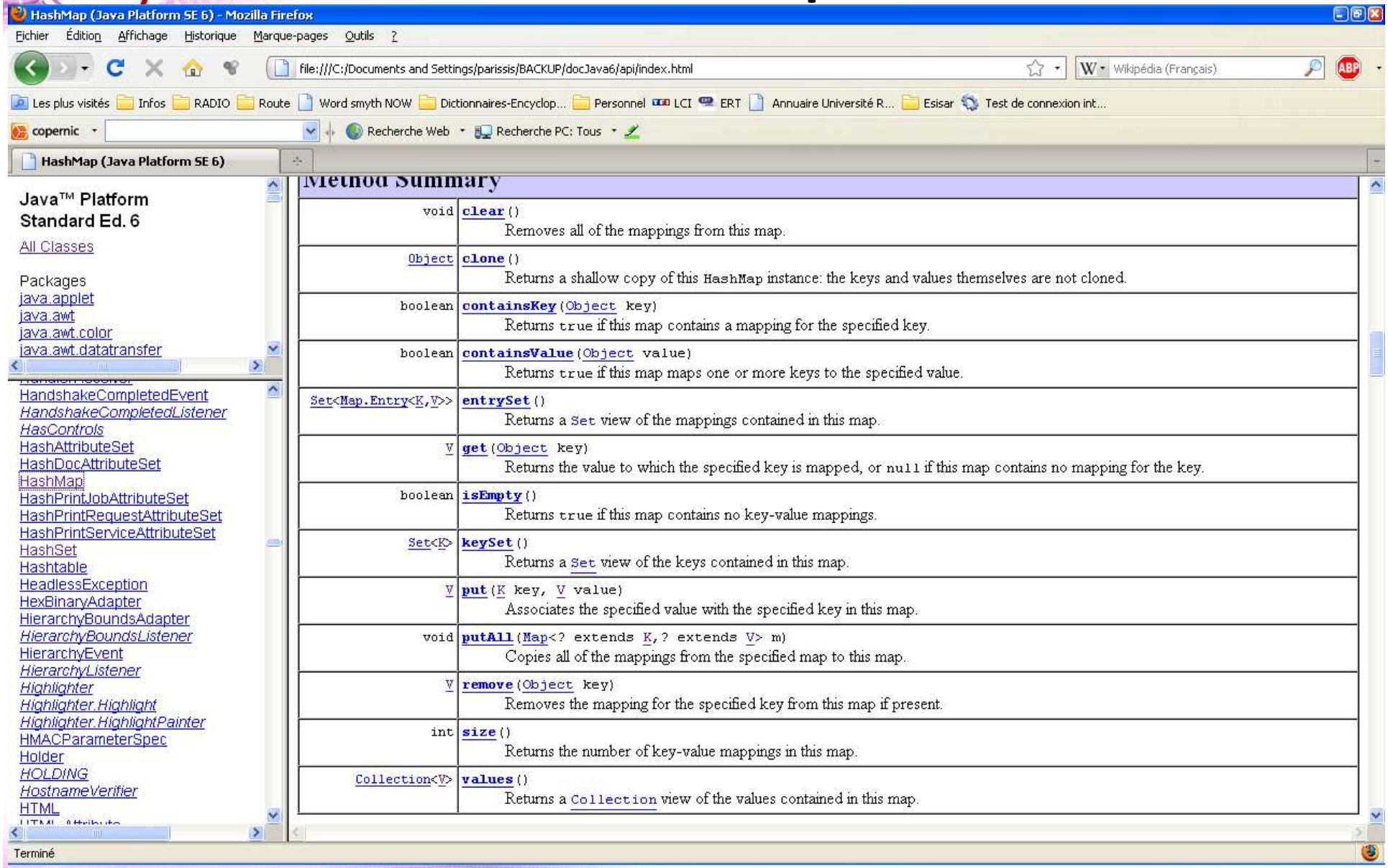

HashSet (Java Platform SE 6) - Mozilla Firefox
Fichier Édition Affichage Historique Marque-pages Outils ?
file:///C:/Documents and Settings/parissis/BACKUP/docJava6/api/index.html
Les plus visités Infos RADIO Route Word smyth NOW Dictionnaires-Encyclop... Personnel LCI ERT Annuaire Université R... Esisar Test de connexion int...
Recherche Web Recherche PC: Tous
HashSet (Java Platform SE 6)
Java™ Platform Standard Ed. 6
All Classes
Packages
java.applet
java.awt
java.awt.color
java.awt.datatransfer
HashAttributeSet
HashDocAttributeSet
HashMap
HashPrintJobAttributeSet
HashPrintRequestAttributeSet
HashPrintServiceAttributeSet
HashSet
Hashtable
HeadlessException
HexBinaryAdapter
HierarchyBoundsAdapter
HierarchyBoundsListener
HierarchyEvent
HierarchyListener
Highlighter
Highlighter.Highlight
Highlighter.HighlightPainter
HMACParameterSpec
Holder
HOLDING
HostnameVerifier
HTML
HTML.Attribute
HTML.Tag
HTML.UnknownTag
HTMLDocument
Constructor Summary
HashSet()
Constructs a new, empty set; the backing HashMap instance has default initial capacity (16) and load factor (0.75).
HashSet(Collection<? extends E> c)
Constructs a new set containing the elements in the specified collection.
HashSet(int initialCapacity)
Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and default load factor (0.75).
HashSet(int initialCapacity, float loadFactor)
Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and the specified load factor.
Method Summary
boolean add(E e)
Adds the specified element to this set if it is not already present.
void clear()
Removes all of the elements from this set.
Object clone()
Returns a shallow copy of this HashSet instance: the elements themselves are not cloned.
boolean contains(Object o)
Returns true if this set contains the specified element.
boolean isEmpty()
Returns true if this set contains no elements.
Iterator<E> iterator()
Returns an iterator over the elements in this set.
boolean remove(Object o)
Removes the specified element from this set if it is present.
int size()
Returns the number of elements in this set (its cardinality).
Terminé

Map, Hashmap : associations

- La classe `Map` définit des collections de paires ***clé/valeur***
- On recherche une valeur en fournissant sa clé
- Exemple: annuaire téléphonique (clé = nom, valeur = téléphone)
- `HashMap` est une implantation particulière de `Map`
- Exemple (Hashmap avec des String comme clés et comme valeurs)

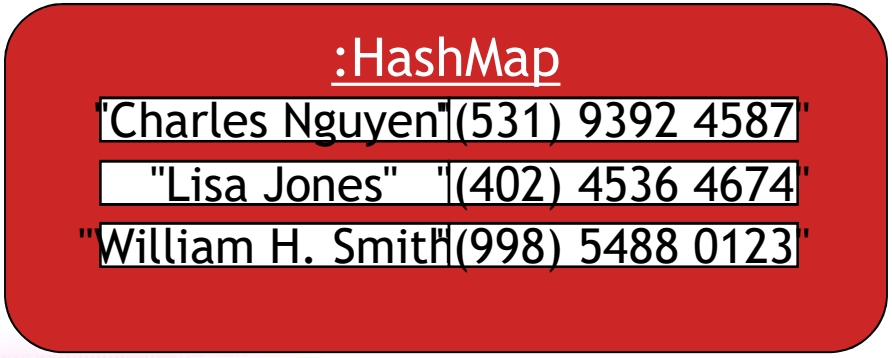


HashMap


HashMap (Java Platform SE 6) - Mozilla Firefox
Fichier Édition Affichage Historique Marque-pages Outils ?
file:///C:/Documents and Settings/parissis/BACKUP/docJava6/api/index.html
Wikipédia (Français)
Les plus visités Infos RADIO Route Word smyth NOW Dictionnaires-Encyclop... Personnel LCI ERT Annuaire Université R... Esisar Test de connexion int...
copernic
Recherche Web
Recherche PC: Tous
HashMap (Java Platform SE 6)
Java™ Platform
Standard Ed. 6
All Classes
Packages
java.applet
java.awt
java.awt.color
java.awt.datatransfer
HandshakeCompletedEvent
HandshakeCompletedListener
HasControls
HashAttributeSet
HashDocAttributeSet
HashMap
HashPrintJobAttributeSet
HashPrintRequestAttributeSet
HashPrintServiceAttributeSet
HashSet
Hashtable
HeadlessException
HexBinaryAdapter
HierarchyBoundsAdapter
HierarchyBoundsListener
HierarchyEvent
HierarchyListener
Highlighter
Highlighter.Highlight
Highlighter.HighlightPainter
HMACParameterSpec
Holder
HOLDING
HostnameVerifier
HTML
Method Summary
void clear() Removes all of the mappings from this map.
Object clone() Returns a shallow copy of this HashMap instance: the keys and values themselves are not cloned.
boolean containsKey(Object key) Returns true if this map contains a mapping for the specified key.
boolean containsValue(Object value) Returns true if this map maps one or more keys to the specified value.
Set<Map.Entry<K,V>> entrySet() Returns a Set view of the mappings contained in this map.
V get(Object key) Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
boolean isEmpty() Returns true if this map contains no key-value mappings.
Set<K> keySet() Returns a Set view of the keys contained in this map.
V put(K key, V value) Associates the specified value with the specified key in this map.
void putAll(Map<? extends K,? extends V> m) Copies all of the mappings from the specified map to this map.
V remove(Object key) Removes the mapping for the specified key from this map if present.
int size() Returns the number of key-value mappings in this map.
Collection<V> values() Returns a Collection view of the values contained in this map.
Terminé

Utilisation de HashMap

```
HashMap <String, String> phoneBook =
              new HashMap<String, String>();

phoneBook.put("Charles Nguyen", "(531) 9392 4587");
phoneBook.put("Lisa Jones", "(402) 4536 4674");
phoneBook.put("William H. Smith", "(998) 5488 0123");


String phoneNumber = phoneBook.get("Lisa Jones");
System.out.println(phoneNumber);
```



Entrées/sorties de texte

- Le paquetage `java.io` fournit des classes pour les E/S
- `java.io.IOException` est une exception sous contrôle.
- Lecteurs (readers), rédacteurs (writers), flux (streams)
  - Les lecteurs et rédacteurs gèrent des textes.
    - Suites de caractères (`char`).
  - Les flux gèrent des données binaires.
    - Suite d'octets (`byte`).

FileWriter

```
try {
    FileWriter writer = new FileWriter("name of file");
    while(there is more text to write) {
        ...
        writer.write(next piece of text);
        ...
    }
    writer.close();
}
catch(IOException e) {
    something went wrong with accessing the file
}
```

FileReader

```
try {
    BufferedReader reader =
        new BufferedReader(new FileReader("filename"));
    String line = reader.readLine();
    while(line != null) {
        do something with line
        line = reader.readLine();
    }
    reader.close();
}
catch(FileNotFoundException e) {
    the specified file could not be found
}
catch(IOException e) {
    something went wrong with reading or closing
}
```

E/S avec le terminal

- System.in
  - `java.io.InputStream`
- `java.util.Scanner`
  - `nextInt`, `nextLine`, etc.
- `Scanner` et `File` peuvent remplacer `BufferedReader` et `FileReader`.