Génie Logiciel: Comment coder pour les autres

Daniel Porumbel
dp.cnam@gmail.com

nombreaux slides dus à
Pierre Cubaud

Plan de l'exposé

1) structurer son application en Java
- Les interfaces
- Les paquetages
- Gestion de la visibilité : public, private etc

2) documenter le code : javadoc

3) la conception du logiciel
- Cycle de production du code
- Diagrammes de classes

4) conclusion générale

1) structurer son code en Java

Modularité 1 : réaliser une longue tache

- Une tache récurrente dans les jeux video consiste à vérifier si deux objets s'intersectent.
- tester si les rectangles ci-dessous (de même taille) s'intersectent
  - $(x1A, y1A, x2A, y2A)$
  - $(x1B, y1B, x2B, y2B)$.

Modularité 1 : réaliser une longue tache

- Une tache récurrente dans les jeux video consiste à vérifier si deux objets s'intersectent.
- tester si les rectangles ci-dessous (de même taille) s'intersectent
  - $(x1A, y1A, x2A, y2A)$
  - $(x1B, y1B, x2B, y2B)$.

Une solution parfaitement fonctionnelle mais mauvaise !

```
if !(((x1A<x1B && x2A>x1B) && (y1A<y1B && y2A>
   y1B)) || ((x1A<x2B && x2A>x2B) && (y1A<y1B
   && y2A>y1B)) || ((x1A<x1B && x2A>x1B) && (
   y1A<y2B && y2A>y2B)) || ((x1A<x2B && x2A>x2B
   ) && (y1A<y2B && y2A>y2B)))
        return false;
    else
        return true;
```

Modularité 2 : décomposition de la tache

Comparer ce code avec :

```
if (pointInt(x1B,y1B,x1A,y1A,x2A,y2A))
    return true;
if (pointInt(x2B,y1B,x1A,y1A,x2A,y2A))
    return true;
if (pointInt(x1B,y2B,x1A,y1A,x2A,y2A))
    return true;
if (pointInt(x2B,y2B,x1A,y1A,x2A,y2A))
    return true;
return false;
```

A retenir : La programmation modulaire permet de décomposer (factoriser) une grosse tache en plusieurs taches plus petites.

Avantages et inconvenients de la modularité

- La modularité rend le code plus lisible !
- La modularité est parfois sacrifiée pour l'efficacité ou pour rendre la compréhension plus difficile (obfuscation)
- Le code peut aussi être généré automatiquement

Exemple : le code `html` du site `www.google.fr`

Avantages et inconvenients de la modularité

- La modularité rend le code plus lisible !
- La modularité est parfois sacrifiée pour l'efficacité ou pour rendre la compréhension plus difficile (obfuscation)
- Le code peut aussi être généré automatiquement

Exemple : le code `html` du site `www.google.fr`

```
<!doctype html><html itemscope="" itemtype="http:
google.j.b=(!!location.hash&&!!location.hash.matc
||(google.j.qbp==1);(function(){google.c={c:{a:tr
function(a,b,c){google.timers[a]||google.startTic
!0};google.c.u=function(a){var b=google.timers.lo
function(a){google.c.c.a&&google.afte&&google.tic
h||b.ctrlKey||b.shiftKey||b.altKey||b.metaKey||H(
B(b),g=(g.type||g.tagName).toUpperCase(),(g=32==(
typeof b[p]&&"srcElement"!==p&&"target"!==p&&(c[p
```

Style moderne : utiliser souvent des classes standard (code déjà-écrit)

Avantage On n'a pas besoin de re-inventer la roue

Avantage Les classes standard sont souvent mieux écrites que nos classes

Style moderne : utiliser souvent des classes standard (code déjà-écrit)

Avantage On n'a pas besoin de re-inventer la roue

Avantage Les classes standard sont souvent mieux écrites que nos classes

Inconvenient on peut sacrifier l'efficacité

Inconvenient on nous impose un **raisonnement assez figé**, **(presque) pensée unique**

- même la classe `String` nous impose une manière particulière de travail : on n'est pas censé changer une lettre d'une chaîne déjà construite

`String[]` est immutable mais `char[]` ne l'est pas.
On va les comparer.

Le Phénomène “mille fonctionnalités”

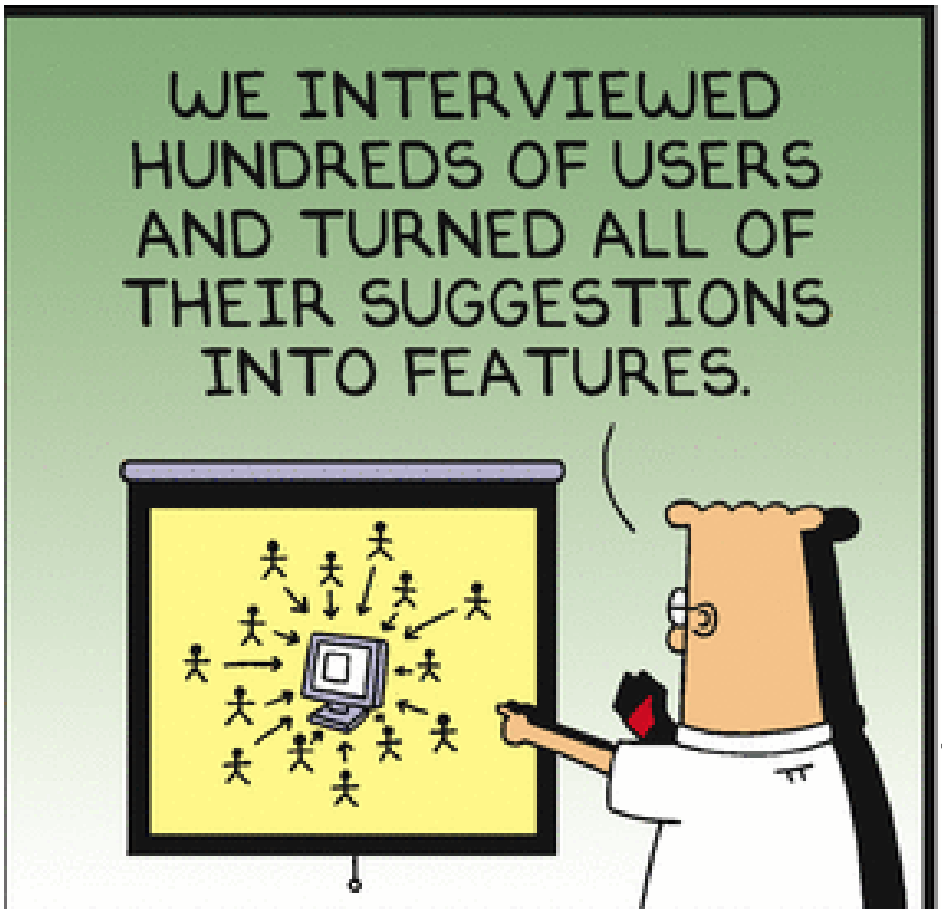
Très souvent on utilise un code efficace qui a été soumis en fait à beaucoup de de contraintes pas forcement nécessaires

- On n’a pas toujours besoin d’un code `thread-safe`

Le Phénomène “mille fonctionnalités”

Très souvent on utilise un code efficace qui a été soumis en fait à beaucoup de de contraintes pas forcement nécessaires

- On n’a pas toujours besoin d’un code `thread-safe`



Nous avons contacté des centaines d’utilisateurs et on a transformé toutes leurs suggestions en fonctionnalités

Résultat : il y a des logiciels avec trop de fonctionnalités et de fenêtres (Windows) qui peuvent noyer l’utilisateur plus que ça ne l’aide.

Règles et bonnes pratiques de programmation

En théorie, il y de nombreuses régles sur Internet :

- *ne pas utiliser de GO TO*
  - Le but est d'éviter la "programmation spaghettis", mais ça existe en C++
  - James Gosling a crée la JVM originale avec le support de GO TO, mais il finalement effacé cette fonctionnalité car il trouve qu'on n'en a pas besoin. `http://stackoverflow.com/questions/2545103/is-there-a-goto-statement-in-java`
- *le nom d'une variable doit commencer par une minuscule*
- *il faut séparer les attributs privés des attributs publiques*
- *il faut écrire des interfaces*
- *ne pas utiliser de variables globales* ...

Sur “ne pas utiliser de variable globale”

L’idée essentielle à retenir est l’objectif général : la clarté. La règle “pas de variable globale” est là uniquement à cause du fait que les variables globales rendent le code moins clair.

Cependant, comme beaucoup de règles, les gens retiennent la règle et non pas le objectif final de la règle.

J’ai vu des programmes qui doublent la taille du code simplement pour éviter le mal des variables globales ;

Tom West, forum `http://stackoverflow.com/questions/484635/are-global-variables-bad`

Pas de “operator overloading” sous Java

En $C++$, le code ci-après permet d’appeller une méthode de la classe Frac ;
Sous Java, il faut écrire `z=x.add(y)`.

```
Frac x, y, z;
z = x + y;
```

Pas de “operator overloading” sous Java

En $C++$, le code ci-après permet d’appeller une méthode de la classe Frac ;
Sous Java, il faut écrire `z=x.add(y)`.

```
Frac x, y, z;
z = x + y;
```

J’ai pris la décision de ne pas utiliser d’`operator overloading` en Java comme un choix personnel parce que j’ai vu trop de gens en abuser sous C++.
I left out operator overloading as a fairly personal choice because I had seen too many people abuse it in C++.

James Gosling. Source : `www.gotw.ca/publications/c_family_interview.htm`

Le `Operator overloading` sous C++

De nombreuses décisions C++ trouvent leurs racines dans mon aversion à forcer les gens faire les choses d'une manière particulière. J'ai souvent été tenté d'interdire des fonctionnalités qui me déplaisaient, mais je me suis abstenu de le faire, parce que je ne pense pas que j'aie le droit d'imposer ma vision sur les autres.

Many C++ design decisions have their roots in my dislike for forcing people to do things in some particular way [...] Often, I was tempted to outlaw a feature I personally disliked, I refrained from doing so because I did not think I had the right to force my views on others.

Bjarne Stroustrup. Source : The Desing and Evolution of C++ (1.3 General Background)

3) documenter son code avec javadoc

```
main(k){float i,j,r,x,y=-16;while(puts(""),y++<15)for(x
=0;x++<84;putchar(" .:-;!/>)|&IH%*#"[k&15]))for(i=k=r=0;
j=r*r-i*i-2+x/25,i=2*r*i+y/10,j*j+i*i<11&&k++<111;r=j);}
```

http://mrl.nyu.edu/~perlin/

Principe de javadoc :

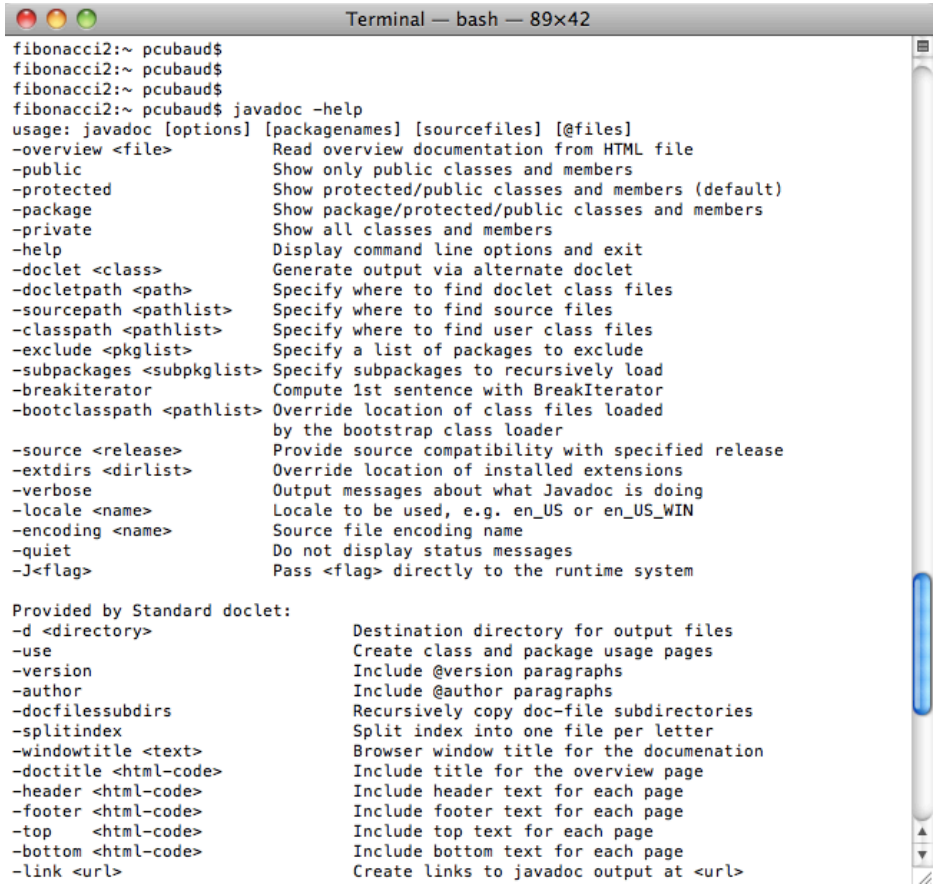
- exploiter les commentaires, avec une astuce de syntaxe :
/** au lieu de /*

- ajouter dedans des balises (tags) à la sémantique spécifique
@author
@version
@see
@param
@return
@exception

- au lieu de compiler avec javac, on compile avec javadoc

- le résultat est un lot de pages HTML, pour lire avec
un navigateur Web

Nombreuses options : javadoc -help



javadoc -private -author -version MonFichier.java

Un exemple (exercice du dernier ED)

```
class MonException extends Exception {}

/**
* Un objet superArray contient juste un tableau d'entier
* avec une méthode permettant de faire des recherches de contenu
*/
class SuperArray {
int[] t;
SuperArray(int dim){
 t = new int[dim];
 for (int i=0;i<t.length;i++) t[i]=i;
}
/**
* Parcours sequentiel du tableau jusqu'à ce que le motif
* donné soit trouvé, ou que la fin du tableau soit rencontrée
* @param r le motif de recherche
*/
int Search(int r) throws MonException {
 boolean succ = false;
 int i = 0;
 while (!succ && i<t.length) {
  succ = (r == t[i] );
  i++;
 }
 if (succ) return i-1;
 else throw new MonException();
}
}
```

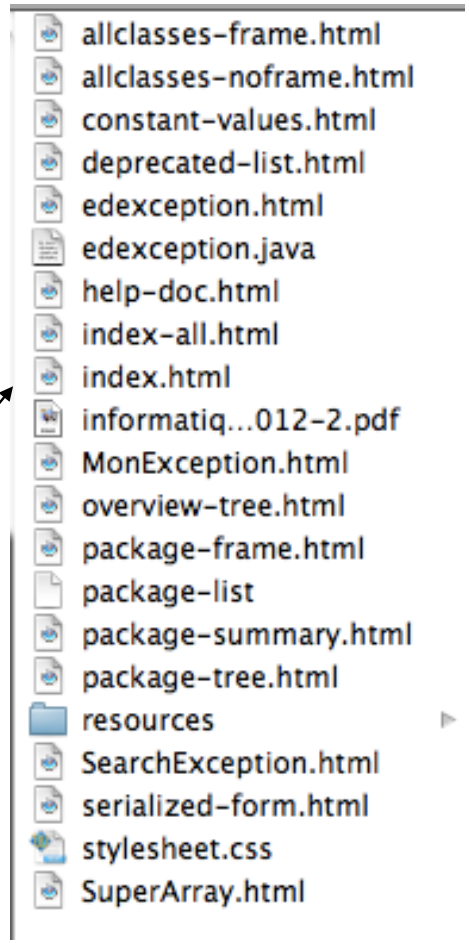
```
/**
* Super programme de demonstration du mécanisme
* des exceptions
* @author Pierre Cubaud
* @version 1.0
*/
class edexception {
 public static void main(String[] arg){
  SuperArray sa = new SuperArray(50);
  int resultat ;
  try {
   resultat = sa.Search(51);
   System.out.println(resultat);
  }
  catch (MonException e){
   System.out.println("echec total lamentable, sorry");
  }

 }
}
```

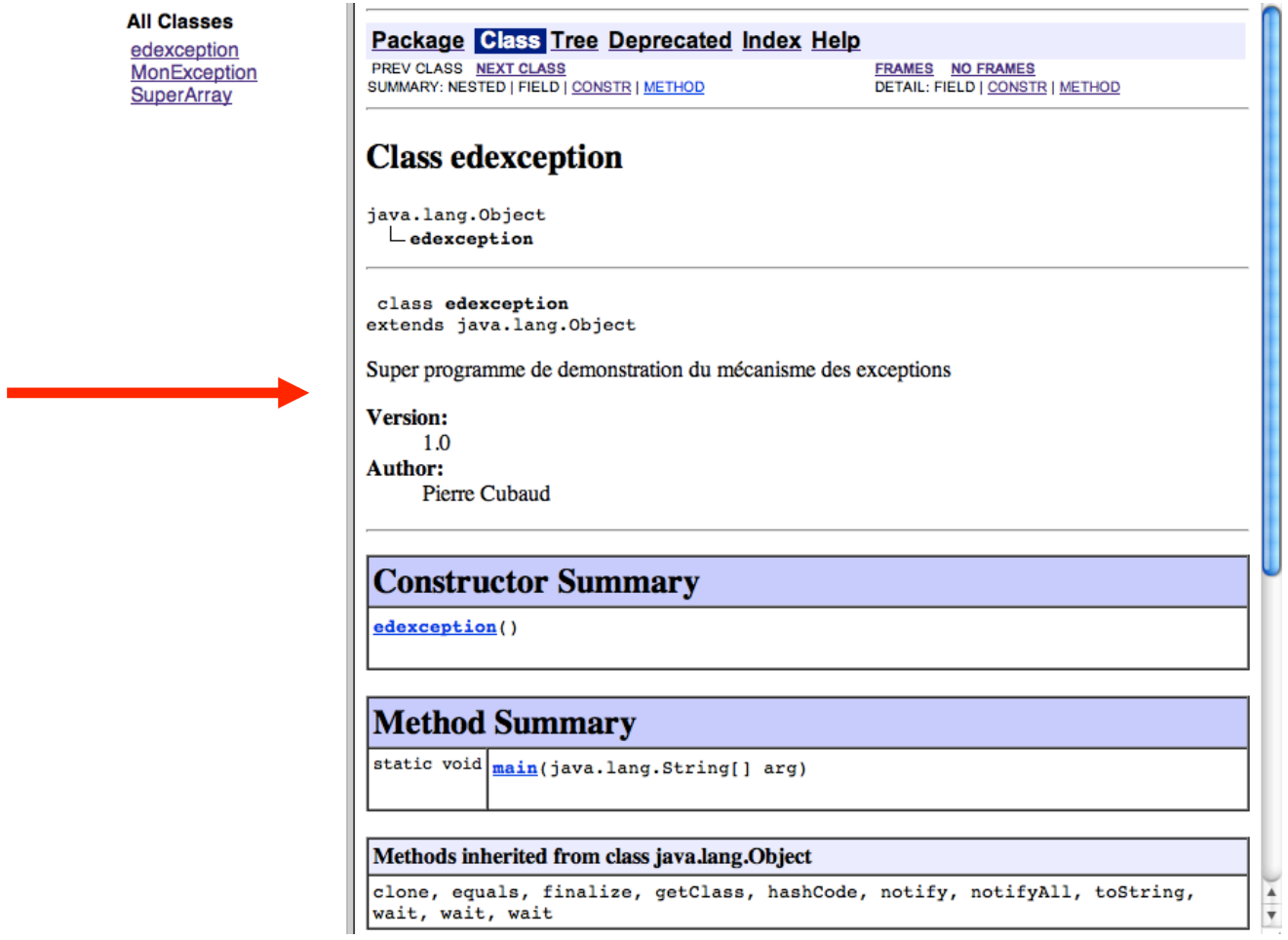
lancement

```
fibonacci2:14 - les progs pcubaud$
fibonacci2:14 - les progs pcubaud$ javadoc -private -author -version edexception.java
Loading source file edexception.java...
Constructing Javadoc information...
Standard Doclet version 1.6.0_17
Building tree for all the packages and classes...
Generating edexception.html...
Generating MonException.html...
Generating SuperArray.html...
Generating package-frame.html...
Generating package-summary.html...
Generating package-tree.html...
Generating constant-values.html...
Building index for all the packages and classes...
Generating overview-tree.html...
Generating index-all.html...
Generating deprecated-list.html...
Building index for all classes...
Generating allclasses-frame.html...
Generating allclasses-noframe.html...
Generating index.html...
Generating help-doc.html...
Generating stylesheet.css...
fibonacci2:14 - les progs pcubaud$
```

résultat : plein
de fichiers HTML



la page d'accueil (index.html)



All Classes
edexception
MonException
SuperArray

Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class edexception

```
java.lang.Object
  └ edexception
```

```
 class edexception
extends java.lang.Object
```

Super programme de demonstration du mécanisme des exceptions

Version:
1.0
Author:
Pierre Cubaud

Constructor Summary

edexception()

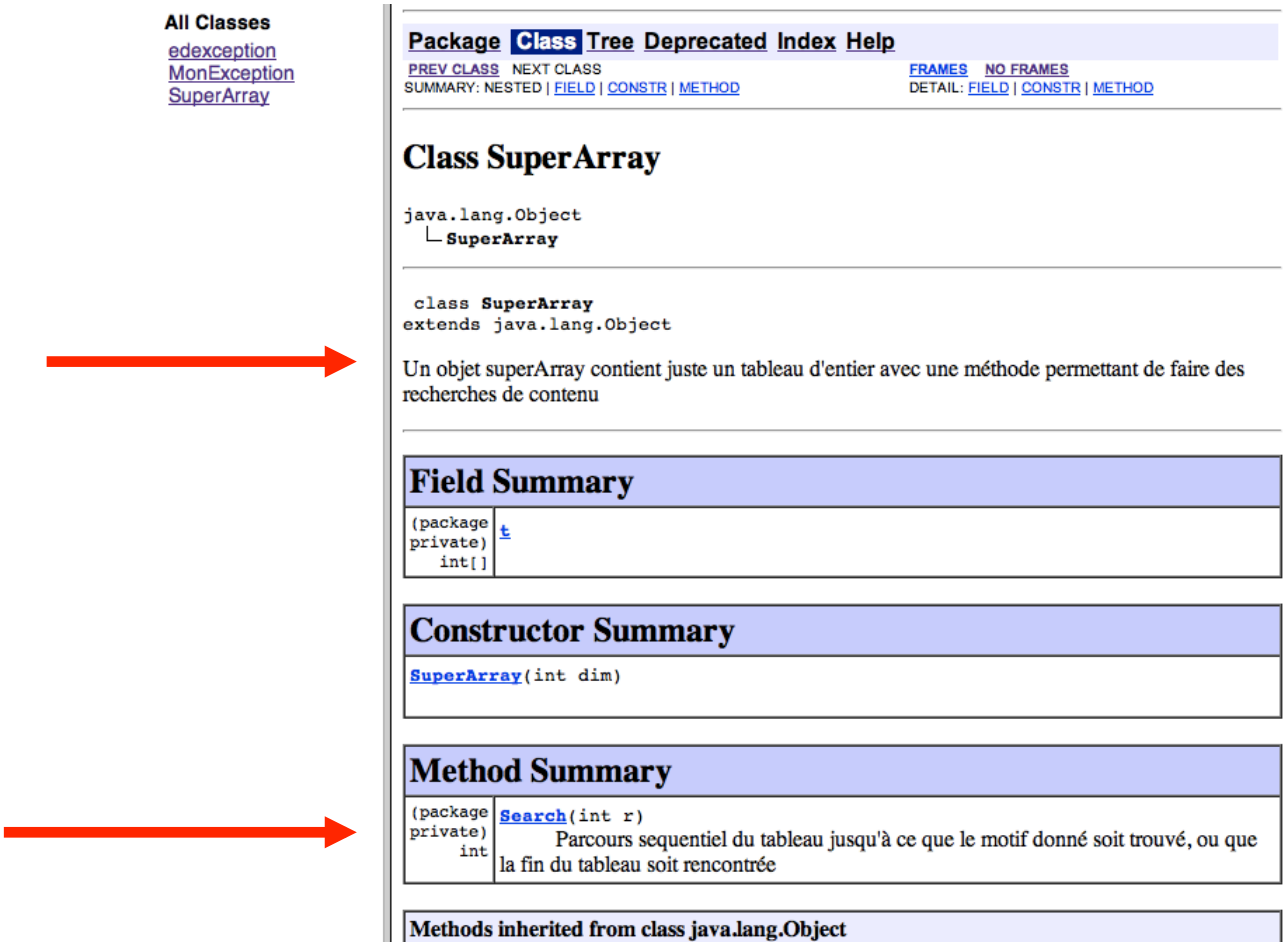
Method Summary

static void	main(java.lang.String[] arg)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Vue pour une classe (Class)



All Classes

edexception
MonException
SuperArray

Package **Class** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS — FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD — DETAIL: FIELD | CONSTR | METHOD

Class SuperArray

```
java.lang.Object
  └ SuperArray
```

```
 class SuperArray
extends java.lang.Object
```

Un objet superArray contient juste un tableau d'entier avec une méthode permettant de faire des recherches de contenu

Field Summary

(package private) int[]	t

Constructor Summary

SuperArray(int dim)

Method Summary

(package private) int	Search(int r) Parcours sequentiel du tableau jusqu'à ce que le motif donné soit trouvé, ou que la fin du tableau soit rencontrée

Methods inherited from class java.lang.Object

suite

All Classes

edexception
MonException
SuperArray

wait, wait, wait

Field Detail

t

```
int[] t
```

Constructor Detail

SuperArray

```
SuperArray(int dim)
```

Method Detail

Search

```
int Search(int r)
    throws MonException
```

Parcours sequentiel du tableau jusqu'à ce que le motif donné soit trouvé, ou que la fin du tableau soit rencontrée

Parameters:

`r` - le motif de recherche

Throws:

MonException

Package **Class** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS — FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD — DETAIL: FIELD | CONSTR | METHOD

NFP136 = NFP135 + 1

Informatique

Accueil > UES > Classements : > IMO > NFP136 : Valeur d'Accueil et de Reconversion en Informatique 2

FORMATIONS
UES
AUDITEURS
VIE PRATIQUE
- Actualités

CONTACTS
Rechercher...

Espace privé (enseignants)

Rss

NFP136 : Valeur d'Accueil et de Reconversion en Informatique 2

- Fiche descriptive de l'UE: Cliquez ici
- Fiche complémentaire de l'UE en présentiel: Cliquez ici
- Emploi du temps de l'UE prévisionnel (sujet à modification): Cliquez ici
- Agenda de l'UE au format iCal: http://deptmedia.cnam.fr/agendas/NFP136.ics
- Diplômes: LG004 Licence STIG, mention informatique générale
- Mots clés: Système UNIX , Compression de données , Algorithmique
- Enseignant responsable: BENTZ Cédric
- **REMPLISSEZ LE FORMULAIRE D'APPRECIATION DES AUDITEURS SUR L'ENSEIGNEMENT**

Responsable national : Cédric Bentz
‣ Enseignants : Cédric Bentz, Amélie Lambert

Précisions pour l'année 2014-2015

‣ Planning de l'UE

Attention : cette UE n'ouvrira pas en FOD cette année !

Supports de cours

‣ Chapitre 1 : Introduction et complexité
‣ Chapitre 2 : Listes, files et piles
‣ Chapitre 3 : Introduction aux graphes
‣ Chapitre 4 : Structures arborescentes
‣ Chapitre 5 : Algorithmes de tri
‣ Chapitre 6 : Algorithmes gloutons
‣ Chapitre 7 : Automates et langages réguliers
‣ Chapitre 8 : Systèmes (partie 1/3)
‣ Chapitre 9 : Systèmes (partie 2/3)
‣ Chapitre 10 : Systèmes (partie 3/3)