Résultat du Sondage concernant JDK 7

par Vincent Brabant (Page d'accueil de Vincent Brabant) (Blog)

Date de publication : 16 décembre 2008

Dernière mise à jour :

Lors de l'édition 2007 de Javapolis, Neal Gafter et Joshua Bloch avaient animé un BOF en fin de soirée pour demander l'avis des utilisateurs Java sur des propositions de modification de langage qui seraient éventuellement introduites dans le JDK 7. Cet article fait le point sur les résultats des sondages que nous avions lancé dans le forum Java de developpez.com.

I - Introduction

Lors de l'édition 2007 de Javapolis, Neal Gafter et Joshua Bloch avaient animé un BOF en fin de soirée pour demander l'avis des utilisateurs Java sur des propositions de modification de langage qui seraient éventuellement introduites dans le JDK 7.

A la suite de cela, nous avions proposé à Neal Gafter de demander votre avis à propos de ces propositions.

L'édition 2008 de Javapolis (renommé en Devoxx) vient de se terminer. Il était donc temps de faire le point sur vos votes et de les faire parvenir auprès de Sun, Neal Gafter et Joshua Bloch, ainsi que les autres Java-Champions.

Cet article a donc pour but de faire le point sur ces propositions, et le résultat des votes.

Nous allons reparcourir chacune des propositions, les illustrer par un petit bout de code, pour finalement terminer pas vos votes.

II - Les différentes propositions de modification de langage

II-A - Proposition 1: Constructeurs simplifiés pour la généricité

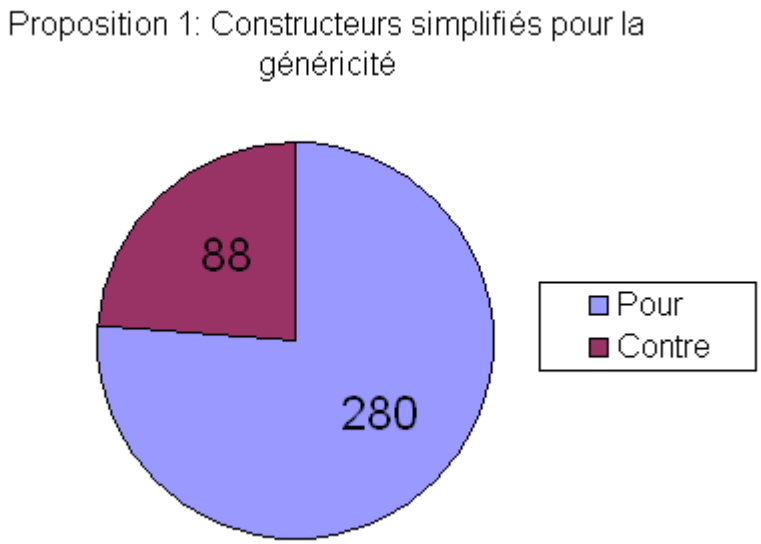
Aujourd'hui :

```
Map<String, List<String>>
  anagrams = new HashMap<String, List<String>>();
```

Demain :

```
Map<String, List<String>>
  anagrams = new HashMap<>();
```

Cette proposition a été vue 8546 fois. Elle a généré et a recueilli 368 votes répartis comme suit : 280 votes pour et 88 votes contre.



Cette proposition a donc été acceptée par 76% des votants.

II-B - Proposition 2 : Créer facilement des collections

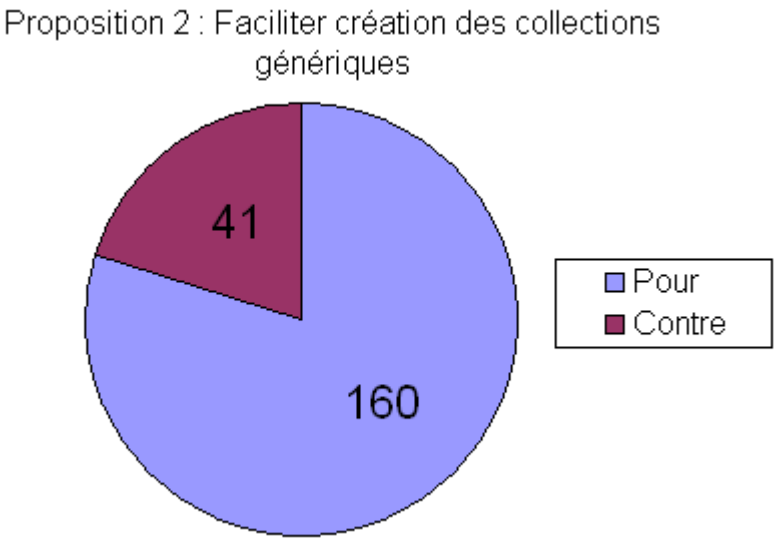
Aujourd'hui :

```
public <E> Set<E> emptySet() { □ }
void timeWaitsFor(Set<Man> people) { □ }

// * Won't compile!
timeWaitsFor(Collections.emptySet());

// OK
timeWaitsFor(Collections.<Man>emptySet());
```

Demain :

```
public <E> Set<E> emptySet() { □ }
void timeWaitsFor(Set<Man> people) { □ }

// OK
timeWaitsFor(Collections.emptySet());
```

Cette proposition a été vue 7141 fois. Elle a généré et a recueilli 201 votes répartis comme suit : 160 votes pour et 41 votes contre.



Cette proposition a donc été acceptée par 80% des votants.

II-C - Proposition 3 : Comparer les énumérations

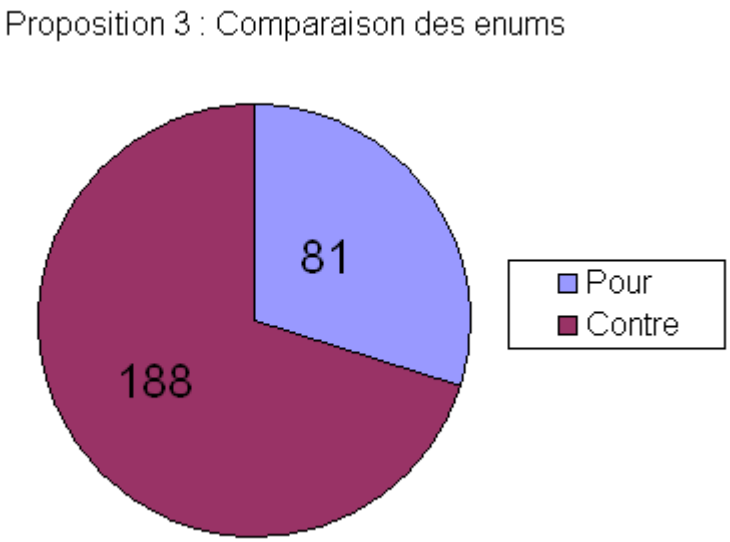
Aujourd'hui :

```
enum Size { SMALL, MEDIUM, LARGE }

if (mySize.compareTo(yourSize) >= 0)
    System.out.println("You can wear my pants.");
```

Demain :

```
enum Size { SMALL, MEDIUM, LARGE }

if (mySize > yourSize)
    System.out.println("You can wear my pants.");
```

Cette proposition a été vue 7891 fois. Elle a généré et a recueilli 269 votes répartis comme suit : 81 votes pour et 188 votes contre.



Cette proposition a donc été rejetée par 70% des votants.

II-D - Proposition 4 : Support de String dans les Switch Case

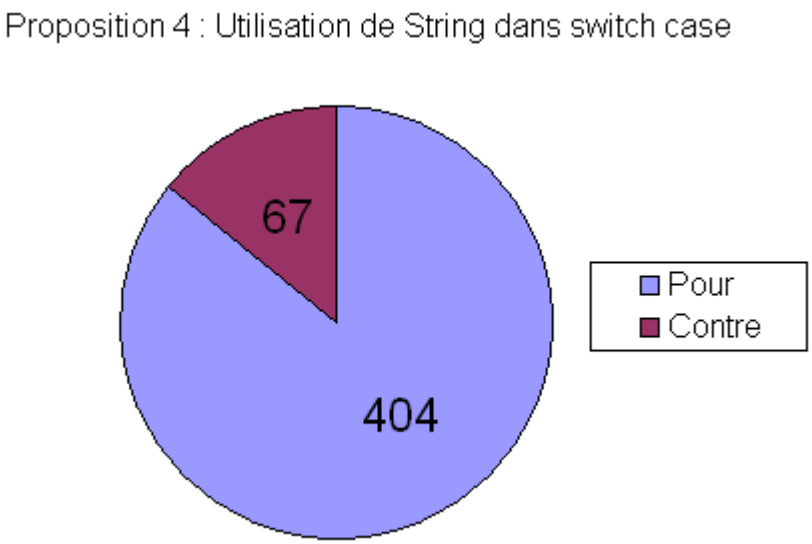
Aujourd'hui :

```
static boolean booleanFromString(String s) {
    if (s.equals("true")) {
        return true;
    } else if (s.equals("false")) {
        return false;
    } else {
        throw new IllegalArgumentException(s);
    }
}
```

Demain :

```
static boolean booleanFromString(String s) {
    switch(s) {
      case "true":
        return true;
      case "false":
        return false;
      default:
        throw new IllegalArgumentException(s);
    }
}
```

Cette proposition a été vue 13706 fois. Elle a généré et a recueilli 471 votes répartis comme suit : 404 votes pour et 67 votes contre.



Cette proposition a donc été acceptée par 86% des votants.

II-E - Proposition 5 : Extensions de méthodes

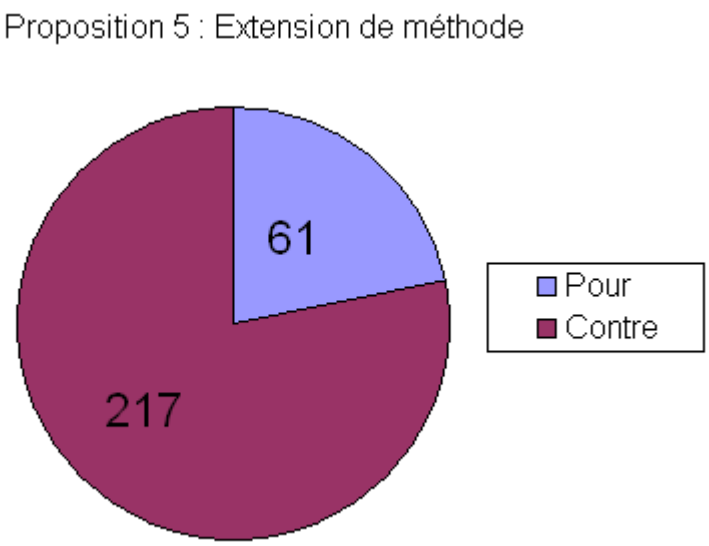
Aujourd'hui :

```
import java.util.Collections;

List<String> list = □;
Collections.sort(list);
```

Demain :

```
import static java.util.Collections.sort;

List<String> list = □;
list.sort();
```

Cette proposition a été vue 8478 fois. Elle a généré et a recueilli 278 votes répartis comme suit : 61 votes pour et 217 votes contre.



Cette proposition a donc été rejetée par 78% des votants.

II-F - Proposition 6 : Invocation chainée

Aujourd'hui :

```
class Builder {
    void setSomething(Something x) { □ }
    void setOther(Other x) { □ }
    Thing result() { □ }
}

Builder builder = new Builder();
builder.setSomething(something);
builder.setOther(other);
Thing thing = builder.result();
```

Demain :

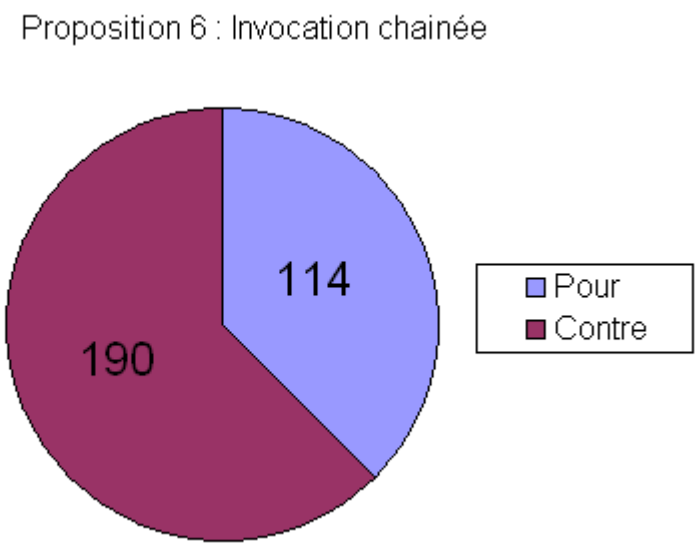
```
class Builder {
    void setSomething(Something x) { □ }
    void setOther(Other x) { □ }
    Thing result() { □ }
}

Thing thing = new Builder()
    .setSomething(something)
    .setOther(other)
    .result();
```

En combinant cette proposition avec la proposition 5, il serait possible d'écrire du code de ce style :

```
import static java.util.Collections.sort;

List<String> strings = ...;
```

```
strings
    .filter(isCountryName) // could be a closure
    .sort()
    .uniq()
    .each(printString); // ditto
```

Cette proposition a été vue 9398 fois. Elle a généré et a recueilli 304 votes répartis comme suit : 114 votes pour et 190 votes contre.



Cette proposition a donc été rejetée par 63% des votants.

II-G - Proposition 7 : catcher plusieurs exceptions

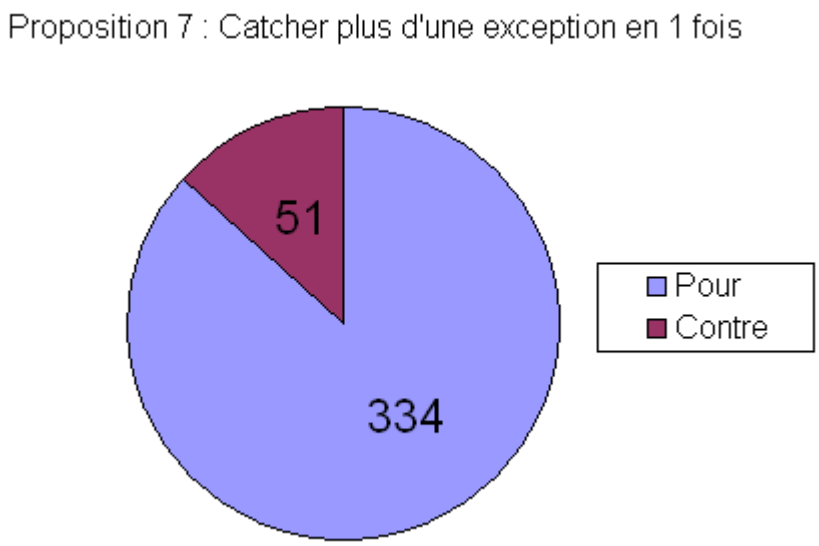
Aujourd'hui :

```
try {
    return klass.newInstance();
} catch (InstantiationException e) {
    throw new AssertionError(e);
} catch (IllegalAccessException e) {
    throw new AssertionError(e);
}
```

Demain :

```
try {
    return klass.newInstance();
} catch (InstantiationException | IllegalAccessException e) {
    throw new AssertionError(e);
}
```

Cette proposition a été vue 12392 fois. Elle a généré et a recueilli 385 votes répartis comme suit : 334 votes pour et 51 votes contre.



Cette proposition a donc été acceptée par 87% des votants.

Mais, si l'on examine les commentaires, vous préfereriez en fait que ce soit la virgule (,) qui soit utilisée pour séparer les exceptions, et non le pipe (|)

II-H - Proposition 8 : rethrow exceptions

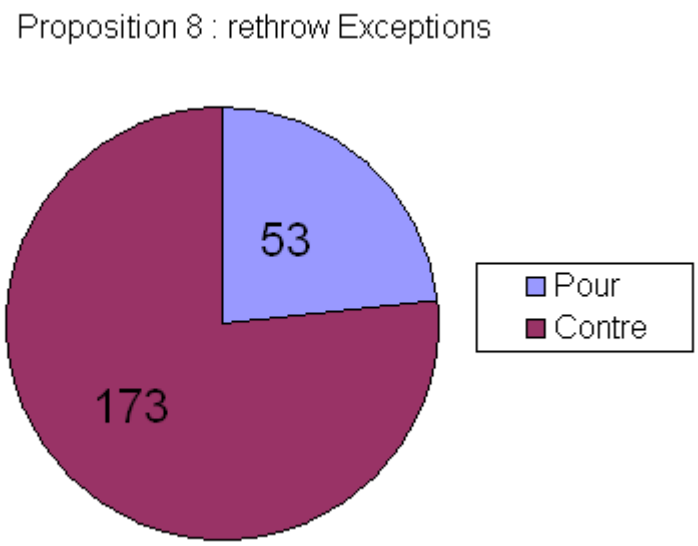
Aujourd'hui :

```
try {
  doable.doit(); // Throws several types
} catch (Throwable ex) {
  logger.log(ex);
  throw ex; // Error: Throwable not declared
}
```

Demain :

```
try {
  doable.doit(); // Throws several types
} catch (final Throwable ex) {
  logger.log(ex);
  throw ex; // OK: Throws the same several types
}
```

Cette proposition a été vue 7122 fois. Elle a généré et a recueilli 226 votes répartis comme suit : 53 votes pour et 173 votes contre.



Cette proposition a donc été rejetée par 77% des votants.

II-I - Proposition 9 : Notation de tableau pour List et Map

Aujourd'hui :

```
void swap(List<String> list, int i, int j) {
    String s1 = list.get(i);
    list.set(i, list.get(j));
    list.set(j, s1);
}
```

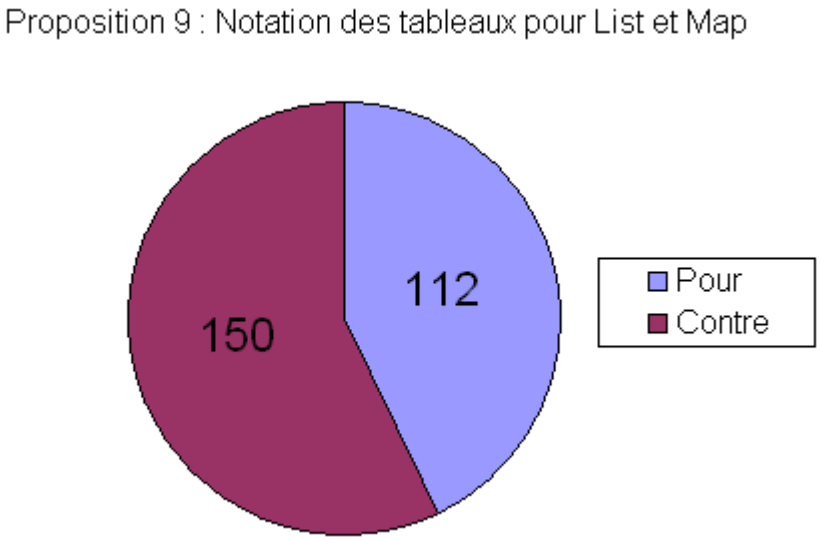
```
Map<Input,Output> cache = □;
Output cachedComputation(Input in) {
    Output out = cache.get(in);
    if (out == null) {
        out = computation(input);
        cache.put(in, out);
    }
    return out;
}
```

Demain :

```
void swap(List<String> list, int i, int j) {
    String s1 = list[i];
    list[i] = list[j];
    list[j] = s1;
}
```

```
Map<Input,Output> cache = □;
Output cachedComputation(Input in) {
    Output out = cache[in];
    if (out == null) {
        out = computation(input);
        cache[in] = out;
    }
    return out;
}
```

Cette proposition a été vue 8105 fois. Elle a généré et a recueilli 262 votes répartis comme suit : 112 votes pour et 150 votes contre.



Cette proposition a donc été rejetée par 57% des votants.

II-J - Proposition 10 : Le même type

Aujourd'hui :

```
class Buffer {
    Buffer flip() { □ }
    Buffer position(int newPos) { □ }
    Buffer limit(int newLimit) { □ }
}
class ByteBuffer extends Buffer {
    ByteBuffer put(byte data) { □ }
}
```

```
ByteBuffer buf = ...;
buf.flip().position(12); // OK
buf.flip().put(12); // Error
((ByteBuffer)(buf.flip())).put(12); // Obliged to cast
```

Demain :

1ière suggestion de syntaxe

```
class Buffer {
    this flip() { □ }
    this position(int newPos) { □ }
    this limit(int newLimit) { □ }
}
class ByteBuffer extends Buffer {
    this put(byte data) { □ }
}
```

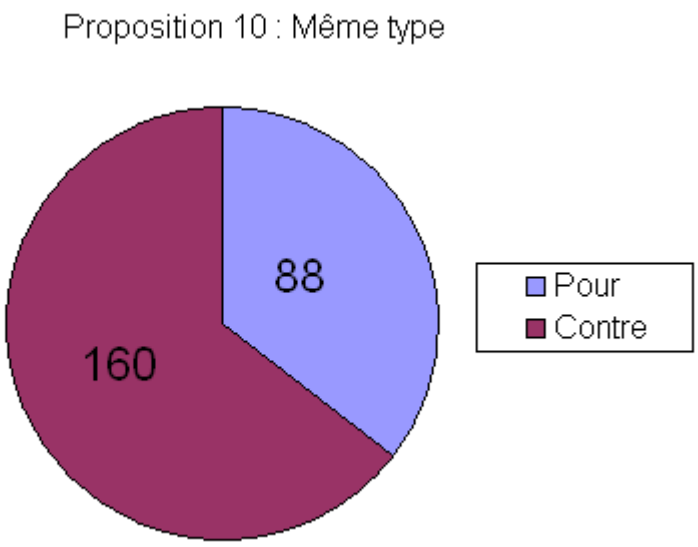
2ième suggestion de syntaxe

```
class Buffer {
    This flip() { □ }
    This position(int newPos) { □ }
    This limit(int newLimit) { □ }
}
class ByteBuffer extends Buffer {
    This put(byte data) { □ }
}
```

Et finallement

```
ByteBuffer buf = ...;
buf.flip().position(12); // OK
buf.flip().put(12); // OK
```

Cette proposition a été vue 8657 fois. Elle a généré et a recueilli 248 votes répartis comme suit : 88 votes pour et 160 votes contre.



Cette proposition a donc été rejetée par 65% des votants.

II-K - Proposition 11 : Typedef

Typedef, c'est permettre de créer une espèce d'alias ou de raccourcis.

Par exemple :

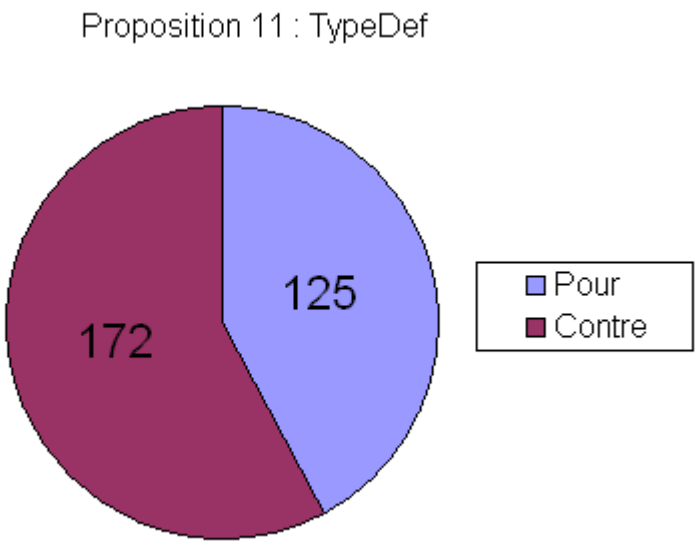
1ière suggestion :

```
import java.util.Map<String,String> as MyProperties;
import java.util.Map<String,T> as Lookup<T>;
// if we add BGGA function types[
import { double => double } as DoubleFcn;
```

2ième suggestion :

```
static class MyProperties = Map<String,String>;
static class Lookup<T> = Map<String,T>;
// if we add BGGA function types
static class DoubleFcn = { double => double };
```

Cette proposition a été vue 8546 fois. Elle a généré et a recueilli 297 votes répartis comme suit : 125 votes pour et 172 votes contre.



Cette proposition a donc été rejetée par 58% des votants.

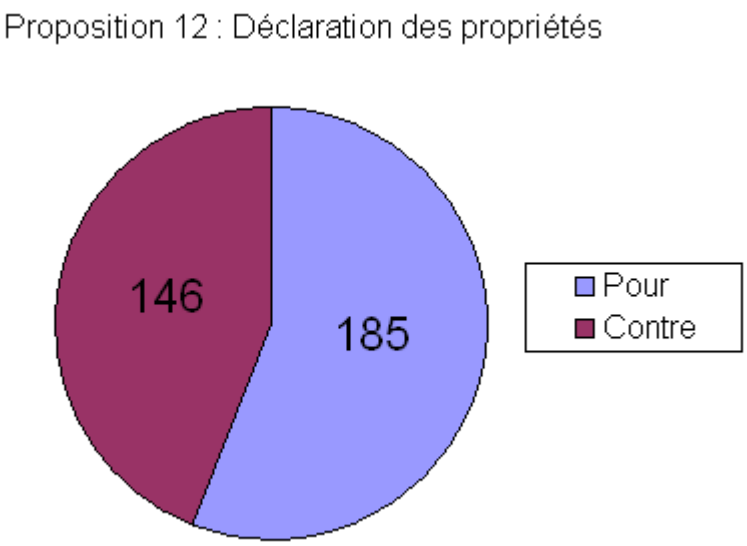
II-L - Proposition 12 : Déclaration des propriétés

Le but de cette proposition est de supprimer le fait d'écrire des setters et getters

Demain :

```
public class Person {
 public property String forename;
 public property int age;
}
```

Cette proposition a été vue 11005 fois. Elle a généré et a recueilli 331 votes répartis comme suit : 185 votes pour et 146 votes contre.



Cette proposition a donc été acceptée par 56% des votants.

II-M - Proposition 13 : Accès aisé aux propriétés

1ière suggestion : En utilisant le point comme séparateur

```
public class Person {
 public property String forename;
 public property int age;
}
Person p = new Person();
p.forename = "Stephen";   // calls setter
String str = p.forename;  // calls getter
```

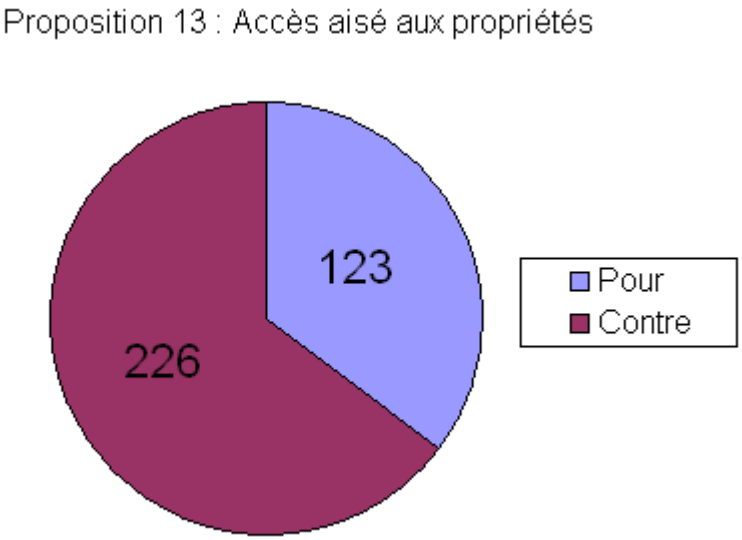
2ième suggestion : En utilisant => comme séparateur

```
public class Person {
 public property String forename;
 public property int age;
}
Person p = new Person();
p=>forename = "Stephen";   // calls setter
String str = p=>forename;  // calls getter
```

3ième suggestion : En utilisant # comme séparateur

```
public class Person {
 public property String forename;
 public property int age;
}
Person p = new Person();
p#forename = "Stephen";   // calls setter
String str = p#forename;  // calls getter
```

Cette proposition a été vue 10257 fois. Elle a généré et a recueilli 349 votes répartis comme suit : 123 votes pour et 226 votes contre.



Cette proposition a donc été rejetée par 65% des votants.

III - Conclusion

III-A - Etude des votes

Parmis ces 13 propositions de modification de langage, vous n'en avez accepté que 5 dont une seule de façon mitigée, les autres ayant plus de 75 % en leur faveur.

Les propositions acceptées, par ordre décroissants de vote total

- Proposition 4 : Le support de String dans le Switch Case (471 votes)
- Proposition 7 : Catcher plusieurs exceptions (385 votes)
- Proposition 1 : Constructeurs simplifiés pour la généricité (368 votes)
- Proposition 12 : Déclaration des propriétés (331 votes)
- Proposition 2 : Créer facilement des collections (201 votes)

La plupart d'entre vous (87%) avez clairement envie de pouvoir faire des Switch Case avec des String.
Et c'est la même majorité (86%) qui désirez pouvoir catcher plusieurs exceptions en une seule fois, mais vous avez tout de même laisser des commentaires pour marquer votre préférence pour la virgule et non le pipe "|" comme séparateur.

Parmis les 8 propositions rejetées, seules 3 le sont avec plus de 75% de votes. (les propositions 5, 3 et 8), les autres variant entre 57% (proposition 9) et 65% (proposition 13).

Les propositions rejetées le sont donc moins fermement que les propositions acceptées

Les propositions rejetées par ordre décroissant de vote total

- Proposition 13 : Accès aisés aux propriétés (349 votes)
- Proposition 6 : Invocation chainée (304 votes)
- Proposition 11 : Typedef (297 votes)
- Proposition 5 : Extensions de méthodes (278 votes)
- Proposition 3 : Comparaison énumération (269 votes)

- Proposition 9 : Notation de tableau pour List et Map (262)
- Proposition 10 : Le même type (248 votes)
- Proposition 8 : rethrow exceptions (226 votes)

Une autre conclusion étrange est que vous avez été 56% à voter pour la proposition 12 qui concerne la déclaration des propriétés dans une classe, mais vous avez été 65% à rejeter les façons d'accéder à ces propriétés.

III-B - Et maintenant ?

Les résultats de ces votes ont été transmis sur la mailing list des Java-Champions, à laquelle sont également abonnés des représentants de Sun.

Nous allons également la transférer à l'employé de Sun qui a fait la keynote sur le JDK 7 lors de l'édition 2008 de Devoxx.

Pour 2009, nous allons lancer de nouveaux sondages, toujours pour le JDK 7, où nous vous demanderons cette fois-ci de choisir des priorités parmis 8 propositions de langage qui ont été retenues par Sun.

Vous remarquerez d'emblée que les 8 propositions de langage retenues par Sun ne sont pas issues uniquement des 13 propositions pour lesquelles vous étiez invité à voter.

Ces nouveaux votes seront tout aussi importants que ceux que vous avez effectués durant cette année 2008, puisque cette fois-ci, vous aurez à donner une priorité sur les différentes propositions. Mais tout cela vous sera expliqué plus en détails ultérieurement.