



Programmation parallèle et distribuée en Java™

Serge Chaumette
LaBRI, Laboratoire Bordelais de Recherche en Informatique
Université Bordeaux I
Serge.Chaumette@labri.u-bordeaux.fr

iHperf2000





Programmation parallèle et distribuée en Java™

Serge Chaumette
LaBRI, Laboratoire Bordelais de Recherche en Informatique
Université Bordeaux I
Serge.Chaumette@labri.u-bordeaux.fr

iHperf2000



2



Programmation parallèle et distribuée en Java™

Serge Chaumette
LaBRI, Laboratoire Bordelais de Recherche en Informatique
Université Bordeaux I
Serge.Chaumette@labri.fr


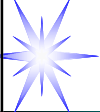
3



Programmation Java™

Serge Chaumette
LaBRI, Laboratoire Bordelais de Recherche en Informatique
Université Bordeaux I
Serge.Chaumette@labri.u-bordeaux.fr

4





Programmation Java™

Approfondissement et introduction aux technologies avancées

Serge Chaumette
LaBRI, Laboratoire Bordelais de Recherche en Informatique
Université Bordeaux I
Serge.Chaumette@labri.u-bordeaux.fr

5


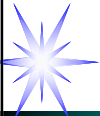


Copyright

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. The author is independent of Sun Microsystems, Inc.

© 2002 -- Serge Chaumette

6





Objectif

Connaître les concepts/technologies clefs intégrées dans Java pour savoir ce qui est possible ou pas.

Être capable de faire le choix Java ou pas.

© 2002 -- Serge Chaumette

7

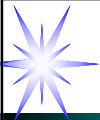


Objectif


Connaître les concepts/technologies clefs intégrées dans Java pour savoir ce qu'on peut faire ou pas, comment le faire, ou et comment se documenter.

© 2002 -- Serge Chaumette

8

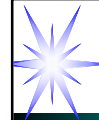


Objectif




1. Approfondissement du langage.
2. Connaître les concepts/technologies clés intégrées dans Java en particulier en termes de programmation dynamique (chargement dynamique, réflexion, sérialisation, *beans*, etc.).

© 2002 -- Serge Chaumette 9



Objectif




Connaître les concepts/technologies clés intégrées dans Java pour savoir ce qui est possible ou pas.


Passer de C++ à Java

Être capable de faire le choix Java ou pas.

© 2002 -- Serge Chaumette 10



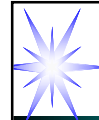
Cohérence des exemples




→ Trois domaines seront régulièrement utilisés pour illustrer le propos :

- ◆ Les véhicules
- ◆ Les formes géométriques
- ◆ Les points, i.e. les couples de valeurs

© 2002 -- Serge Chaumette 11




Plan




- Les langages à objets
- L'environnement Java
- Java et le *Web*
- Le langage Java

© 2002 -- Serge Chaumette 12



Plan (suite)



- Les *threads*
- La communication par *sockets*
- *Serialisation* Invocation de méthodes distantes
- Codes (agents) mobiles
- Jini
- CORBA
- La réflexion

© 2002 -- Serge Chaumette 13

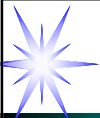


Les langages à objets




- Abstraction
 - ◆ Classes
 - ◆ Objets
 - ◆ Instanciation
- Encapsulation
- Communication par messages
- Héritage

© 2002 -- Serge Chaumette 14



[Les langages à objets / Abstraction]

Classes



```

classe Maison

    int nombreDePieces;

    Maison(int nombreDePieces){
        this.nombreDePieces=nombreDePieces;
    }
    boolean plusGrandeQue(Maison autreMaison){


        return this.nombreDePieces >
            autreMaison.nombreDePieces;
    }
  
```

© 2002 -- Serge Chaumette 15



[Les langages à objets / Abstraction]

Objets et instanciation



```

Maison maisonDePierre = new Maison(4);
Maison maisonDePaul = new Maison(5);
  
```

© 2002 -- Serge Chaumette 16

[Les langages à objets]

Communication par messages

boolean comparaison =
maisonDePierre.plusGrandeQue(maisonDePaul);

© 2002 -- Serge Chaumette 17

[Les langages à objets]

Héritage

```

classe MaisonAvecJardin hérite de Maison

double tailleJardin;

MaisonAvecJardin(int nombre DePieces, int tailleJardin){
    super(nombreDePieces);
    this.tailleJardin=tailleJardin;
}

double superficieDuJardin(){
    return tailleJardin;
}
    
```

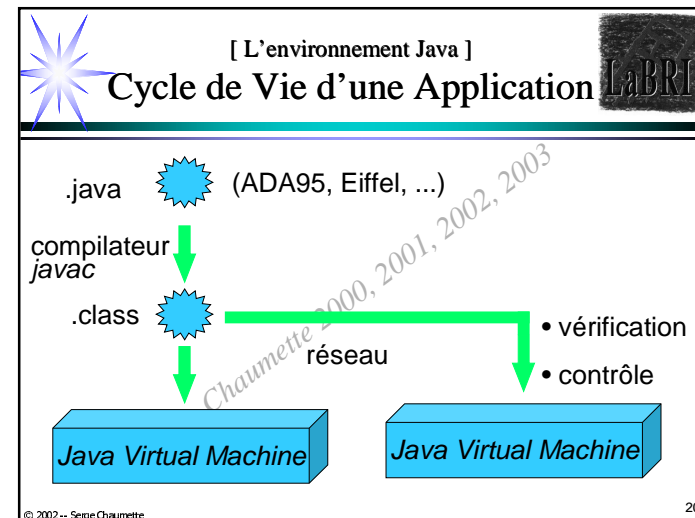
© 2002 -- Serge Chaumette 18

[L'environnement Java]

L 'environnement Java

- Cycle de vie d'une application
- Environnement de base
- Java côté source
- Java côté *byte-code*
- Les outils Java
- Evolution de la technologie

© 2002 -- Serge Chaumette 19



[L'environnement Java]
Environnement de base

Java Development Kit (JDK)

- un compilateur → *javac*
- un interpreteur ou machine virtuelle Java → *java*
- un outil de visualisation d'*applets* → *appletviewer*

© 2002 -- Serge Chaumette 21

[L'environnement Java]
Java côté source

- Syntaxe proche de C++
- Mécanisme d'exceptions
- Paquetages
- Documentation intégrée

© 2002 -- Serge Chaumette 22

[L'environnement Java / Java côté source]
Exemple

```

public class Château extends Object {
    private int surface_;
    ....
    public int surface() {
        return surface_;
    }
    public int quantité(int année) {
        ....
        ....
    }
}

```

© 2002 -- Serge Chaumette 23

[L'environnement Java]
Java côté *byte-code*

- Java est compilé en *byte-code* :
 - ◆ indépendant de toute plate-forme
 - ◆ qui s' exécute sur une **machine virtuelle**
- Le *byte-code* est sûr :
 - ◆ aucun accès direct à la mémoire
 - ◆ références symboliques
 - ◆ entièrement vérifiable (selon Sun)

© 2002 -- Serge Chaumette 24

[L'environnement Java / Java côté *byte-code*]

Exemple

```

class Château extends java.lang.Object {
    public int surface();
    public int quantité(int);
    Château();

    Method int surface()
    0 aload_0
    1 getfield #3 <Field Château.surface_ I>
    4 ireturn
}

```

© 2002 -- Serge Chaumette 25

[L'environnement Java / Java côté *byte-code*]

Exploitations possibles

- Cible possible pour un compilateur
 - ◆ Portabilité du code généré
 - ◆ Mécanismes intégrés
- Java temps réel
 - ◆ Jeu d'instructions connu : possibilité de faire des mesures
- Migration sûre

© 2002 -- Serge Chaumette 26

[L'environnement Java / Les outils Java]

Java Workshop

© 2002 -- Serge Chaumette 27


[L'environnement Java / Les outils Java]

Java Studio

© 2002 -- Serge Chaumette 28

[L'environnement Java]

Evolution de la technologie




- ◆ Base
 - ◆ *Java Development Kit (JDK)*
 - ◆ *APIs*
- ◆ *JavaOS*
 - ◆ sur un système existant
 - ◆ *stand alone*
- ◆ Processeurs Java
 - ◆ *JavaPC*

© 2002 -- Serge Chaumette 29

[L'environnement Java]

Evolution de la technologie (suite)




- ◆ Java embarqué
 - ◆ *J2ME*
 - ◆ Téléphones, PDAs
- ◆ JavaCard
- ◆ Composants
 - ◆ *JavaBeans*
 - ◆ *Jini*
 - ◆ ...

© 2002 -- Serge Chaumette 30

[L'environnement Java]

Java et le Web




- ➔ Principe de fonctionnement
- ➔ Intérêt de la combinaison
- ➔ Mise en œuvre

© 2002 -- Serge Chaumette 31

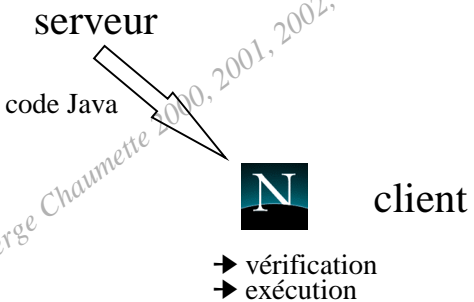
[Java et le WEB]

Principe de fonctionnement



serveur

code Java




client

- ➔ vérification
- ➔ exécution

© 2002 -- Serge Chaumette 32

[Java et le WEB]

Intérêt de la combinaison



- Multi plates-formes
- Exécution côté client
 - ♦ interactivité
 - ♦ coût supporté par le client

© 2002 -- Serge Chaumette 33

[Java et le Web]

Mise en œuvre



Dans un fichier HTML :


exemple.html

```
<APPLET CODEBASE=http://www.labri.u-bordeaux.fr/lib/java/  
        CODE=Demo.class WIDTH=100 HEIGHT=100>  
<PARAM NAME="age" VALUE="18">  
</APPLET>
```

© 2002 -- Serge Chaumette 34

[Java et le WEB]

Le langage Java




- Structure d'un programme
- Exemple
- Compilation et exécution
- Les types
- Le contrôle de flot
- Objets et classes
- Les paquetages
- Les adaptateurs

© 2002 -- Serge Chaumette 35

[Java et le Web]

Le langage Java




- Héritage
- Compatibilité et conversions de type
- Interfaces
- Les exceptions
- Le ramasse-miettes
- Le paquetage IO

© 2002 -- Serge Chaumette 36

[Le langage Java]

Structure d'un programme



```

package labri.tutoriel;
import labri.java.util.*;
class <uneclasse>{
    ...
}
public class <classe>{


```

La classe publique <classe> doit être définie dans le fichier <classe>.java (c'est la seule classe publique de ce fichier)

© 2002 -- Serge Chaumette 37

[Le langage Java]

Exemple



Bonjour.java

```


public class Bonjour{
    public static void main(String args[]){
        System.out.println("Bonjour !");
    }
}

```

© 2002 -- Serge Chaumette 38

[Le langage Java]

Compilation et exécution



Compilation

```
$ javac Bonjour.java
$
```


Exécution

```
$ java Bonjour
Bonjour !
$
$ java -verbose Bonjour
[ ... ]
```

© 2002 -- Serge Chaumette 39

[Le langage Java]

Illustration du chargement dynamique



Bonjour1.java

```

public class Bonjour1{
    public static void main(String args[]){
        System.out.println("Bonjour !");
        new java.awt.Frame();
    }
}

```

© 2002 -- Serge Chaumette 40

[Le langage Java]

Compilation et exécution

Compilation

```
$ javac Bonjour1.java
$
```

Exécution

```
$ java -verbose Bonjour1
[ ... ]
$
```

© 2002 -- Serge Chaumette 41

[Le langage Java]

Les types

→ Primitifs

- ♦ boolean, byte, short, char, int, long, float, double

→ Références

- ♦ En Java il n'y a pas de pointeurs
- ♦ En Java tout est objet

```
public class Demo{
...
    Integer i;
...
}
```

© 2002 -- Serge Chaumette 42

[Le langage Java]

Les types

→ Chaque type primitif est aussi décrit par une *classe* portant le même nom que lui

→ Exemple :

- ♦ int est décrit par la classe Integer
- ♦ Integer.MAX VALUE et Integer.MIN VALUE représentent les valeurs min et max d'un entier

© 2002 -- Serge Chaumette 43

[Le langage Java / Les types]

Le type char

→ En Java les caractères sont codés en UNICODE 2.1 → caractères 16 bits

→ Par exemple

```
int Σ;
float Π;
Téléphone monTéléphone;
```

sont des déclarations de variables correctes

© 2002 -- Serge Chaumette 44

[Le langage Java / Les types]
Le type char

- *Universal character encoding standard*
- Représentation du texte pour un traitement informatisé
- Supporte l'aspect multilingue : tous les caractères utilisés par toutes les langues écrites du monde
- 1 caractère = 1 nom = 1 valeur numérique

© 2002 -- Serge Chaumette 45

[Le langage Java / Les types]
Le type char

- Maintenant, 3 formes de codage
 - ◆ Répertoire commun de caractères
 - ◆ Plus d'1 million de caractères possibles
- BMP
 - ◆ Basic Multilingual Plane
 - ◆ 64k caractères
 - ◆ 8000 possibilités encore disponibles dans le BMP
- 917.476 autres possibilités

© 2002 -- Serge Chaumette 46

[Le langage Java / Les types]
Le type char

- Usage privé
 - ◆ 6.400 possibilités dans le BMP
 - ◆ 131.068 autres possibilités

© 2002 -- Serge Chaumette 47

[Le langage Java / Les types]
Le type char

- Trois formes de codage sont reconnues par le Consortium Unicode pour implémenter le standard :
 - ◆ UTF-8
 - ◆ Suite d'octets
 - ◆ Les caractères ASCII sont inchangés
 - ◆ UTF-16
 - ◆ La plupart des caractères les plus utilisés sont sur 16 bits
 - ◆ Les autres sont sur une paire de 16 bits
 - ◆ Compromis efficacité d'accès/utilisation mémoire
 - ◆ UTF-32
 - ◆ Tous les caractères sont sur 32 bits
 - ◆ Accès rapide/utilisation mémoire importante

© 2002 -- Serge Chaumette 48

[Le langage Java / Les types]

Le type char

→ Toute implémentation de Java doit supporter les codages suivants :

- ◆ US-ASCII : ASCII 7 bits
- ◆ ISO-8859-1 : ISO-Latin 1 8 bits
- ◆ UTF-8
- ◆ UTF-16
- ◆ ...

→ Quand un environnement Java lit un fichier source, il le convertit à la volée en UNICODE.

© 2002 -- Serge Chaumette 49

[Le langage Java / Les types]

Le type char

→ Ceci offre une abstraction du jeu de caractères de la plate-forme locale

→ La classe Character fournit des méthodes pour demeurer indépendant du jeu de caractères :

```
public static boolean isWhiteSpace(char c);
public static boolean isUpperCase(char c);
public static char toUpperCase(char c);
```

© 2002 -- Serge Chaumette 50

[Le langage Java / Les types]

Le type char

→ String

- ◆ La chaîne créée utilise Unicode

→ public byte[] getBytes()

- ◆ Retourne des caractères dans le codage par défaut de la plate-forme courante

→ public byte[] getBytes(String codage)

- ◆ Retourne des caractères dans le codage indiqué

© 2002 -- Serge Chaumette 51

[Le langage Java / Les types]

Le type String

→ String est le type des chaînes de caractères

→ C'est en fait un type référence avec un support syntaxique intégré au langage

→ On peut écrire

- ◆ String s = "exemple de chaîne";

© 2002 -- Serge Chaumette 52

[Le langage Java / Les types / Le type String]

Fonctionnalités principales

- Rappel : c'est un type référence
- Méthodes


```
public char charAt(int i);
public int length();
...
```
- Cf. la documentation de l'API

© 2002 -- Serge Chaumette 53

[Le langage Java / Les types]

Le type tableau

- C'est un type référence avec un support syntaxique intégré au langage
- Déclaration


```
<type>[] <nom du tableau> = new <type>[<dimension>];
<type>[] <nom du tableau> = <initialisation>;
```
- Utilisation


```
<nom de tableau>[<index>]
```

© 2002 -- Serge Chaumette 54

[Le langage Java / Les types]

Le type tableau : exemples

```
int [][]matrice1 = new int[3][10];

int [][]matrice2 = new int[3][];
matrice2[0]=new int[10];
matrice2[1]=new int[10];
matrice2[2]=new int[10];

int [][]matrice3 = { {1}, {1, 2, 0}, {-3, 4}, };
```

© 2002 -- Serge Chaumette 55

[Le langage Java / Les types]

Le type tableau : fonctionnalités principales

- Rappel : c'est un type référence
- Attribut length
- Vérification des bornes : si on déborde du tableau alors l'exception IndexOutOfBoundsException est levée.

© 2002 -- Serge Chaumette 56

[Le langage Java]

Le contrôle de flot

→ while
 → for
 → if/else
 → switch
 → do/while
 → break, continue

→ pas de goto

© 2002 -- Serge Chaumette 57

[Le langage Java]

Le contrôle de flot : break

→ break

- ◆ Met fin à la boucle la plus interne (switch, for, while, ou do)
- ◆ Ne peut apparaître que dans une boucle

→ break label:

- ◆ Met fin à une instruction étiquetée par label:
- ◆ Différent du goto
 - ◆ Pas n'importe où
 - ◆ Protection contre les modifications de code

© 2002 -- Serge Chaumette 58

[Le langage Java]

Le contrôle de flot : continue

→ continue

- ◆ Transfert le contrôle à la fin de la boucle (for, while, ou do)

→ continue label:

- ◆ Transfert le contrôle à la fin de la boucle étiquetée par label:

© 2002 -- Serge Chaumette 59

[Le langage Java]

Objets et classes

- Définition
- Exemple
- Création / destruction
- Les champs
- Les méthodes
- Construction
- Contrôle d'accès
- Modificateurs de classe

© 2002 -- Serge Chaumette 60

[Le langage Java / Objets et Classes]

Définition

→ Un objet

- ♦ est une valeur
- ♦ a un type : c'est sa classe ; il en est une instance

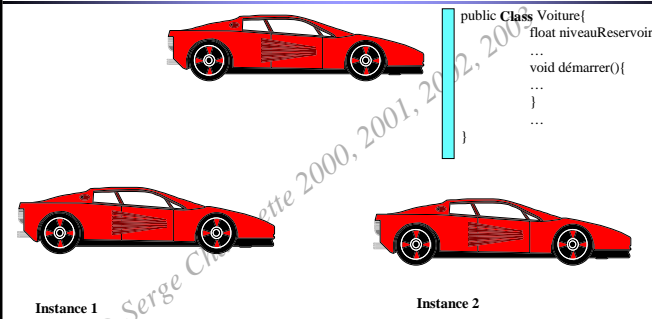
→ Une classe

- ♦ est un type, une description:
 - ♦ Valeurs, état : des champs
 - ♦ Opérations, comportement : les méthodes

© 2002 -- Serge Chaumette 61

[Le langage Java / Objets et Classes]

Exemple



```
public class Voiture{
    float niveauReservoir;
    ...
    void démarrer(){
    ...
    }
    ...
}
```

Instance 1 Instance 2

© 2002 -- Serge Chaumette 62

[Le langage Java / Objets et Classes]

Création/destruction

→ new

- ♦ Les instances sont allouées dans le tas
- ♦ Le type obtenu est un type référence
- ♦ Chaque instance a une copie des champs

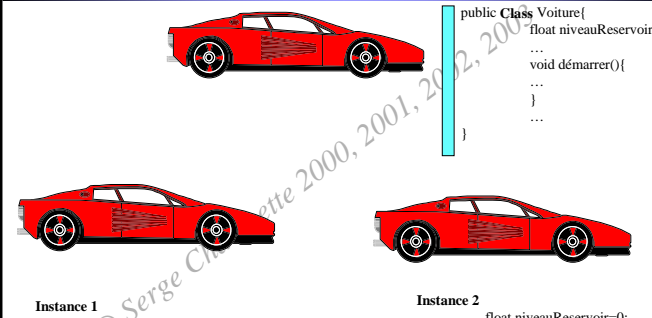
```
Stack maPile = new java.util.Stack();
```

→ Pas de delete mais un ramasse-miettes (*garbage collector*)

© 2002 -- Serge Chaumette 63

[Le langage Java / Objets et Classes]

Les champs



```
public class Voiture{
    float niveauReservoir;
    ...
    void démarrer(){
    ...
    }
    ...
}
```

Instance 1 (float niveauReservoir=100; Instance 2 float niveauReservoir=0;

© 2002 -- Serge Chaumette 64

[Le langage Java / Objets et Classes / Champs]

Champs d'instance, champs statiques

- Champ standard, d'instance
 - ♦ Une copie par instance
- Champ statique ou de classe
 - ♦ Une seule copie dans la classe
- Exemple
 - ♦ le champ `MAX_VALUE` de `java.lang.Integer`

© 2002 -- Serge Chaumette 65

[Le langage Java / Objets et Classes]

Les méthodes

- Les méthodes sont les opérations associées à un type ; elles permettent en particulier l'encapsulation
- On les appelle de la façon suivante :
 - ♦ `<référence d'un objet>.<nom de méthode>(<liste de paramètres>)`
 - ♦ La référence `this` vers l'objet lui-même

© 2002 -- Serge Chaumette 66

[Le langage Java / Objets et Classes / Méthodes]

Méthodes statiques

- Elle est associée à une classe et pas à une instance
- Elle est donc invoquée indépendamment de toute instance et ne nécessite pas la création d'une instance
- Elle ne peut accéder qu'aux éléments statiques de la classe.
- Exemple :
 - ♦ `Integer.parseInt`

© 2002 -- Serge Chaumette 67

[Le langage Java / Objets et Classes / Méthodes]

Surcharge de méthodes

- Surcharger une méthode c'est donner une autre méthode de même nom avec des paramètres et éventuellement un type de retour différent
- En Java, on ne peut pas surcharger les opérateurs
- En Java, on ne peut pas surcharger juste en changeant le type de retour

© 2002 -- Serge Chaumette 68

[Le langage Java / Objets et Classes / Construction]

Les constructeurs

→ C'est la méthode invoquée pour initialiser une instance lors de sa création

- ♦ Par défaut
 - <nom de la classe>()
 - Ne fait rien
- ♦ Définition par le programmeur
 - Le constructeur vide n'existe plus
 - On peut utiliser this pour invoquer un autre constructeur

© 2002 -- Serge Chaumette 69

[Le langage Java / Objets et Classes / Construction]

Blocs d'initialisation

→ Un bloc d'initialisation est un bloc de code exécuté comme s'il se trouvait au début de chaque constructeur

→ Exemple

→ Exécution dans l'ordre

```
public class ExempleBlocInit{
    int i;
    {
        i=10;
    }
    ...
}
```

© 2002 -- Serge Chaumette 70

[Le langage Java / Objets et Classes / Construction]

Initialisation statique

→ Un bloc d'initialisation statique est un bloc de code exécuté lors du chargement de la classe

→ Exemple

→ Exécution dans l'ordre

```
public class ExempleBlocInit{
    static int i;
    static {
        i=10;
    }
    ...
}
```

© 2002 -- Serge Chaumette 71

[Le langage Java / Objets et Classes]

Contrôle d'accès

→ private

- ♦ Uniquement dans sa classe
- ♦ Applicable aux membres seulement

→ package

- ♦ Dans son propre paquetage

→ protected

- ♦ Sous classes et paquetages
- ♦ Applicable aux membres seulement

→ public

- ♦ Partout

© 2002 -- Serge Chaumette 72

[Le langage Java / Objets et Classes]

Modificateurs de classe

- public
 - ◆ Étend l'accessibilité au delà de son propre paquetage
 - ◆ Accessible de partout
- abstract
 - ◆ Classe incomplète présentant des méthodes abstraites
- final
 - ◆ Classe ne pouvant être étendue par des sous-classes
- strictfp

© 2002 -- Serge Chaumette 73

[Le langage Java]

Les paquetages

Les paquetages permettent de regrouper des classes constituant un ensemble cohérent.

Exemple :
java.lang.net regroupe toutes les classes utilisées pour la programmation réseau.

© 2002 -- Serge Chaumette 74

[Le langage Java / Les paquetages]

Les paquetages de base

- *java.lang*
- *java.net*
- *java.rmi*
- *java.awt*
- *java.sql*
- ...

© 2002 -- Serge Chaumette 75

[Le langage Java / Les paquetages / Exemple]

Création d' un paquetage

On définit la classe *Registre* dans le paquetage *labri.java.util*

```

Registre.java
package labri.java.util;

public class Registre{
...
}

```

© 2002 -- Serge Chaumette 76

[Le langage Java / Les paquetages / Exemple]

Utilisation d'un paquetage

On utilise la classe `Registre` de la façon suivante

```
TestRegistre.java
import labri.java.util.Registre;

public class TestRegistre{
...
    Registre registre=new Registre();
}
```

© 2002 -- Serge Chaumette 77

Les adaptateurs des types de base

- Principe
- Exemple : le type `Integer`
- Hiérarchie des *wrappers*
- Méthodes standards

© 2002 -- Serge Chaumette 78

[Le langage Java / Les adaptateurs]

Principe

- En Java tout n'est pas objet
 - ◆ `int`
 - ◆ `boolean`
 - ◆ `float`
- Intérêt des *wrappers*
 - ◆ fonctionnalités sur le type
 - ◆ réification

© 2002 -- Serge Chaumette 79

[Le langage Java / Les adaptateurs]

Exemple : le type `Integer`

```
public final class java.lang.Integer extends java.lang.Number{
    public static final int MIN_VALUE;
    public static final int MAX_VALUE;
    ...
    public int intValue();
    public java.lang.Integer(int);
    ...
}
```

La valeur primitive contenue dans le wrapper est immuable (pas de méthode pour la changer)

© 2002 -- Serge Chaumette 80

[Le langage Java / Les adaptateurs]

Hierarchie des wrappers

```

graph TD
    Object --> Boolean
    Object --> Character
    Object --> Number
    Object --> Void
    Object --> Class
    Number --> Byte
    Number --> Short
    Number --> Integer
    Number --> Long
    Number --> Float
    Number --> Double
  
```

© Serge Chaumette 2000, 2001, 2002, 2003

© 2002 -- Serge Chaumette 81

[Le langage Java / Les adaptateurs]

Méthodes standards

- ◆ Public static Type valueOf(String str);
- ◆ Public String toString();
- ◆ Public Type typeValue();
- ◆ Public int compareTo(Type other);
- ◆ Public int compareTo(Object obj);
- ◆ Public boolean equals(Object obj);
- ◆ Public int hashCode();

© Serge Chaumette 2000, 2001, 2002, 2003

© 2002 -- Serge Chaumette 82

[Le langage Java]

Héritage

- Principe
- Restrictions et syntaxe
- Exemple
- Les constructeurs
- Surcharge
- Redéfinition
- Masquage de champs
- Compatibilité et conversion de types

© Serge Chaumette 2000, 2001, 2002, 2003

© 2002 -- Serge Chaumette 83

[Le langage Java / Héritage]

Principe

© Serge Chaumette 2000, 2001, 2002, 2003

© 2002 -- Serge Chaumette 84

[Le langage Java / Héritage]
Principe (suite)

```

    graph TD
      vehicule --> individuel
      vehicule --> commun
  
```

© 2002 -- Serge Chaumette 85

[Le langage Java / Héritage]
Restrictions et syntaxe

Restrictions

- pas d'héritage multiple
- pas de surcharge ni redéfinition d'opérateurs
- surcharge et redéfinition de méthodes autorisées

Syntaxe

```

class <classe fille> extends <classe mère> {
  ...
}
  
```

© 2002 -- Serge Chaumette 86

[Le langage Java / Héritage]
Exemple

Rectangle.java

```

public class Rectangle{
  private int largeur, hauteur;

  public Rectangle(int largeur, int hauteur){
    this.largeur=largeur;
    this.hauteur=hauteur;
  }
}
  
```

© 2002 -- Serge Chaumette 87

[Le langage Java / Héritage]
Exemple (suite)

Carre.java

```

public class Carre extends Rectangle{

  public Carre(int largeur){
    super(largeur, largeur);
  }
}
  
```

© 2002 -- Serge Chaumette 88

[Le langage Java / Héritage]

Les constructeurs

→ Le constructeur d'une classe invoque nécessairement un constructeur de sa super-classe.

→ La première opération d'un constructeur est soit :

- `this(...)`
- `super(...)`
- Tout autre instruction : dans ce cas `super()` est automatiquement invoqué

© 2002 -- Serge Chaumette 89

[Le langage Java / Héritage]

Surcharge

→ Rappel : surcharger une méthode, c'est définir une autre méthode avec :

- Le même nom
- Des types de paramètres différents

→ On peut surcharger une méthode héritée

© 2002 -- Serge Chaumette 90

[Le langage Java / Héritage]

Redéfinition

→ Redéfinir une méthode, c'est donner une autre implémentation de la méthode

→ La nouvelle méthode a donc :

- Le même nom
- Les mêmes types de paramètres

© 2002 -- Serge Chaumette 91

[Le langage Java / Héritage]

Masquage de champs

→ Un champ dans une sous-classe *masque* le champ de même nom de la super-classe.

→ Il n'y a pas de notion de surcharge pour les champs

© 2002 -- Serge Chaumette 92

[Le langage Java]
Compatibilité et conversion de types

- Conversion implicite
 - ♦ type → super type → super super type → ... --> super
super ... super type → class Object
 - ♦ On parle de *upcasting*
- Conversion explicite
 - ♦ D'un super-type vers un sous-type
 - ♦ (*sous-type*) <référence de super-type>
 - ♦ On parle de *downcasting*

© 2002 -- Serge Chaumette 93

[Le langage Java]
Compatibilité et conversion de types (suite)

© 2002 -- Serge Chaumette 94

[Le langage Java]
Compatibilité et conversion de types (suite)

Exemple: voiture/voiture de ma femme

© 2002 -- Serge Chaumette 95

[Le langage Java]
Interfaces

- Une interface permet de décrire des fonctionnalités communes de divers objets sans en faire une classe.
 - ♦ généricité
 - ♦ héritage multiple (?)
- Une classe peut implémenter plusieurs interfaces.

© 2002 -- Serge Chaumette 96

[Le langage Java]

Différence entre classes et Interfaces

→ Une classe décrit

- ♦ un type
- ♦ son implémentation
- ♦ pas d'héritage multiple d'implémentation

→ Une interface décrit

- ♦ un type
- ♦ elle se limite aux aspects conception
- ♦ héritage multiple de conception

© 2002 -- Serge Chaumette 97

[Le langage Java / Interfaces]

Exemple

Facturable.java

```
public interface Facturable{
    public String reference();
    public double prix();
}
```

© 2002 -- Serge Chaumette 98

[Le langage Java / Interfaces]

Exemple

Facture.java

```
public class Facture{
    ...
    public void ajouter(Facturable facturable){
        ...
        total=total+facturable.prix();
    }
    ...
}
```

© 2002 -- Serge Chaumette 99

[Le langage Java / Interfaces]

Exemple

Voiture.java

```
public class Voiture implements Facturable{
    ...
    public double prix(){
        return valeur;
    }
    public String reference(){
        return constructeur + ``/`` + modele;
    }
    ...
}
```

© 2002 -- Serge Chaumette 100