Base de programmation Objet en JAVA

Frédéric Vernier
(Université Paris-Sud / LRI / LIMSI-CNRS)
Frederic.Vernier,@limsi.fr

Ce cours reprend en grande partie le matériel pédagogique mis au point par Jérôme Nobécourt, Christian Jacquemin et Claude Barras pour l'enseignement de Java en 2001 et 2002 en FIIFO et celui trouvé sur Internet
(http://www.laltruiste.com/, etc.)

Remerciements

- Ce cours reprend en grande partie le matériel pédagogique mis au point par **Jérôme Nobécourt, Christian Jacquemin et Claude Barras** pour l'enseignement de Java en 2001 et 2002 en FIIFO et celui trouvé sur Internet (http://www.laltruiste.com/, etc.)
- Soutien matériel : Mitsubishi Electric et Logitech

Introduction

- Java est un langage de programmation
  - Syntaxe, mots-clés, etc. ORIENTÉ OBJET
- Java est un environnement de programmation
  - Outils de compilation, de paquetage, de tests, etc.
- Java est un standard
  - Sun (premier), mais aussi IBM, Microsoft (C#)
  - Père fondateur : James Gosling
- Java propose un ensemble de bibliothèque standard

Plan 1: Introduction

1. Langage Objet
2. Java
3. Java : Principe De Base
4. Versions
5. Outils du J2SDK
6. Avantages de Java ?
7. Inconvénients de Java ?
8. Mon Premier Programme (1)
9. Mon Premier Programme (2)
10. Mon Premier Programme (3)
1. Mon Second Programme (1)
2. Mon Second Programme (2)
3. Syntaxe de Java
4. Documentation
5. Exemple de documentation
6. Exécution (1)
7. Exécution (2)

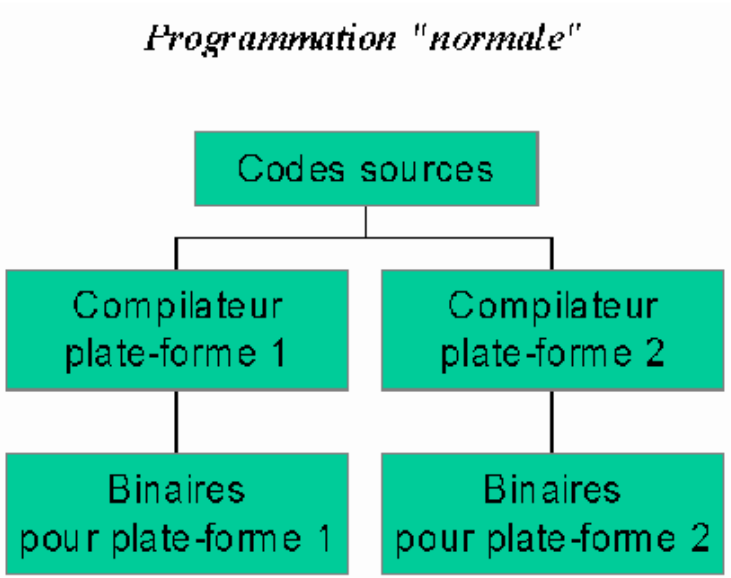
Langage Objet

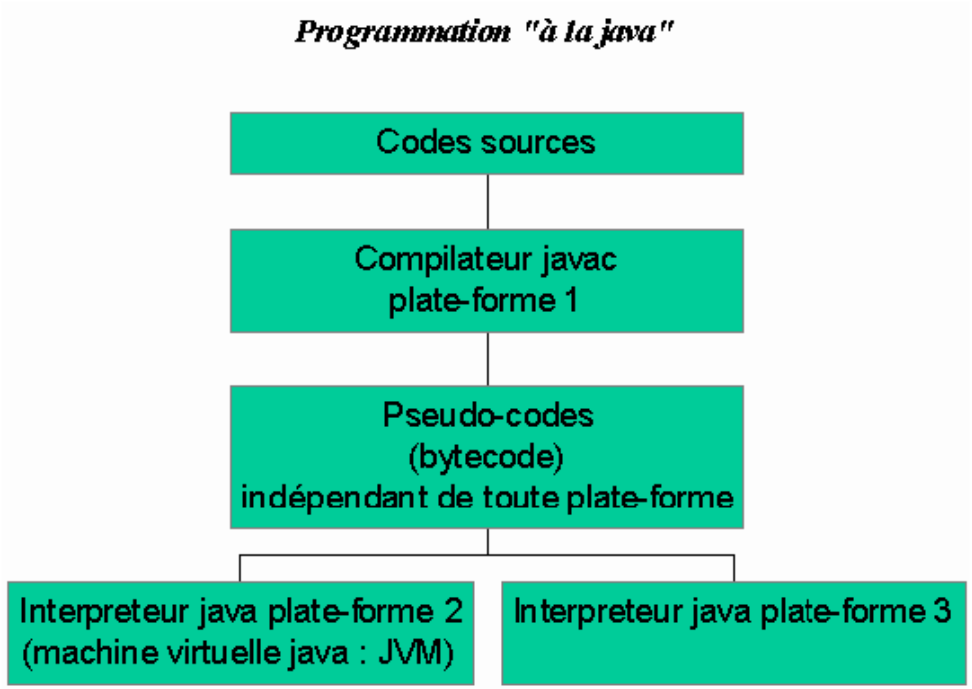
- Langages objets : 1969 (Alan Kay) et Simula (Ole Dahl et Kristen Nygaard)
- Smalltalk
- C : C++
- Pascal : Pascal Objet, Delphi (Borland)
- Basic : VBA (Microsoft)
- C : Objective C
- Eiffel ...

Java

- Créé par Sun (fin 1995)
- James Gosling (initiateur)
- Programmation simple
- Utilisation pour Internet
- Sur la plupart des plate-formes (Archi + OS)
- Gratuit (téléchargeable sur java.sun.com)
- Portable
- Logo (tasse à café), Mascotte (Duke)

Java : Principe de base





En réalité : Plate-forme de référence (Solaris) puis portage sur les autres plate-formes

Versions

- JDK : Java Development Kit
- Java 1.0.2 : 1996, version minimale des browsers web.
  - 212 classes, 8 paquetages
- Java 1.1.5 : 1997
  - 504 classes, 23 paquetages
  - amélioration interface utilisateur (AWT), gestion des erreurs, cohérence du langage
- Java 1.2 : 1998 (JDK 2.0)
  - 1520 classes, 59 paquetages
  - Swing, Drag and drop, amélioration audio
- Java 1.3 : 2001
  - amélioration de la gestion mémoire, rapidité
  - J2SDK (nouveau nom) pour J2EE (Entreprise, serveur), J2SE (app, applet), J2ME (PDA)
- Java 1.4 : 2002
  - XML, expressions régulières, nouvelles E/S…
  - Accès à la mémoire vidéo (VolatileImage) pour l'accélération
- Upcoming 1.5 : Generic (List<String>), foreach, autoboxing and varargs

Outils du J2SDK

- Téléchargeable sur le site java.sun.com
- Outils
  - java : JVM, interpréteur pour les programmes java
  - javac : Compilateur Java
  - appletviewer : JVM pour l'exécution des applets
  - jar : Création et manipulation d 'archive java
  - javadoc : Générateur de documentation Java au format HTML
  - javap : désassembleur de classes Java compilées
  - jdb : débogueur Java en ligne de commande
  - javah : génère le header C ou C++ pour créer un pont compatible entre java et C/C++
- Documentation
  - en ligne : http://java.sun.com/docs/
  - Téléchargeable pour installation en local

Avantages de Java ?

- Ecrire une fois, exécuter partout
- Sécurité
- Gestion automatique de la mémoire
- Réseaux, interfaces, son, pont avec les bases de données en natif
- Exécution dans un navigateur Web (Applet)
- Programmation modulaire et dynamique
- Internationalisation
- Lisibilité du code
- Petitesse du code (tout est dans la JVM)

Inconvénients de Java ?

- Interprété : inefficacité du code
- Nécessite une JVM pour fonctionner
- Gestion de la mémoire inefficace dans la plupart des cas
- Moins de mécanismes objet que C++ pourtant plus ancien (héritage multiple et templates)

Mon premier Programme (1)

```
public class Rien {
}// Fin class Rien
```

- Le fichier java DOIT se nommer : Rien.java
- Il décrit un objet qui s'appelle « Rien »
- Compilation :

```
> javac Rien.java
>
```

- Exécution : On exécute l'objet « Rien » ???

```
> java Rien
In class Rien: void main(String argv[]) is not defined
```

- Message d'erreur : On sait ce qu'il faut faire !

Mon premier Programme (2)

```
public class RienAFaire {
   public static void main(String[] arguments)
   {
   }
}// Fin class RienAFaire
```

- Compilation :

```
> javac RienAFaire.java
>
```

- Exécution :

```
> java RienAFaire
>
```

Attention à la syntaxe Objet !!!

- RienAFaire est une classe sans données
- La fonction main() est une fonction commune à toute les instances
- main() peut même être appelée sans qu'aucune instance soit créée : RienAFaire.main("2")
- RienAFaire != Rien => Noms différents

Mon premier Programme (3)

>javac -verbose RienAFaire.java

```
[parsing started RienAFaire.java]
[parsing completed 248ms]
[loading /home/jacquemi/usr/share/jdk1.3.1_01/jre/lib/rt.jar(java/lang/Object.class)]
[loading /home/jacquemi/usr/share/jdk1.3.1_01/jre/lib/rt.jar(java/lang/String.class)]
[checking RienAFaire]
[wrote RienAFaire.class]
[total 988ms]
```

>java -verbose RienAFaire

```
[Loaded java.lang.Object from /home/jacquemi/usr/share/jdk1.3.1_01/jre/lib/rt.jar]
[Loaded java.io.Serializable from /home/jacquemi/usr/share/jdk1.3.1_01/jre/lib/rt.jar]
[Loaded java.lang.Comparable from /home/jacquemi/usr/share/jdk1.3.1_01/jre/lib/rt.jar]
... (Plus de 200 lignes pour la JVM)
[Loaded java.security.cert.Certificate from /home/jacquemi/usr/share/jdk1.3.1_01/jre/lib/rt.jar]
[Loaded RienAFaire]
[Loaded java.lang.Shutdown$Lock from /home/jacquemi/usr/share/jdk1.3.1_01/jre/lib/rt.jar]
```

Mon second Programme (1)

```
public class Bonjour {
    public static void main (String[] arguments) {
        System.out.println("bonjour");
    }
}// Fin class Bonjour
```

- Compilation :

```
> javac Bonjour.java
>
```

- Exécution :

```
> java Bonjour
bonjour
>
```

Attention à la syntaxe Objet !!!

- System est une classe avec des fonctions de classe (toujours pas besoin d'instance)
- out est une instance de quelque chose qui se trouve être une donnée de l'objet System (voir doc. de System)
- println est une fonction de l'instance out

Mon second Programme (2)

```
public class Bonjour2 {
   public static void main (String[] arguments) {
      System.out.println("bonjour, voici vos arguments : ");
      for (int i=0; i<arguments.length;i++)
         System.out.println(i + " : " + arguments[i]);
   }
}// Fin class Bonjour2
```

- Compilation :

```
> javac Bonjour2.java
```

- Exécution :

```
> java Bonjour2 1 deux 3.14
bonjour, voici vos arguments :
0 : 1
1 : deux
2 : 3.14
```

Attention à la syntaxe Java !!!

- Pas besoin d'une variable (argc) pour connaître la taille d'un tableau
- Boucle for = comme en C
- Construction de la chaîne de caractère avec le signe +

Syntaxe de Java

- Reprend celle du C
- Le ; marque la fin d'instruction
- Les identificateurs sont comme en C (if, while, switch, etc.) et utilisé tel quel
- Les caractères sont codés en UNICODE
  - 16 bits au lieu de 16 => accents dans le code!
  - Les commentaires en mauvais anglais sont préférables à un français impeccable
- Un bloc d'instructions est délimité par {}

Code

```
int maxi = 30;

// Double loop to compute the syrchon factor k
for (int j=0; j<maxi; j++) {
  for (int i=j;i>=0; i--) {
    int k = i+j;

    // compare the syrchon factor k to the Trici sum
    if (k*k/2>i*i+j*j) {
      System.out.println("i="+i+", j="+j+" k="+k);
      System.out.println("i="+i);
    }
    else
      System.out.println("i="+i);
  }
}
```

Code

```
int maxi = 30;

/**
 * Compute the fibi number
 * @param the integer to stop fibi calculation
 * @return the computed fibi number.
**/
public int Fibi(int k_Arg) {
  int k_ret, i = 0;

  // loop to build the Fibi number by summing the first
  // integers
  while (k_ret<k_Arg) {
    k_ret += i;
    i++;
  }
  return k_ret;
}
```

Conventions de Codage

- Les développeurs écrivent du code que d'autres lisent
  - Java est apprécié pour sa lisibilité
  - Sun PROPOSE une façon d'écrire le code
- Conventions de codage: java.sun.com/docs/codeconv/
- Aide au déboguage de Java (communauté open source)
  - Re-formateur de code sinon… (JavaStyle, Jalopy, etc.)

```
class Sample extends Object {
    int ivar1;
    int ivar2;

    Sample(int i, int j) {
        ivar1 = i;
        ivar2 = j;
    }

    int emptyMethod() {}
    ...
}
```

Attention aux petits détails

- Nom de classe avec majuscule
- Nom de fonction avec minuscule
- Indentation

Documentation

Format : Les commentaires dans le fichier .java doivent commencer par /**, finir par */ et chaque ligne doit commencer par une *

Il existe des étiquettes de champs telles que le nom d'auteur qui sont utilisées pour générer une API.

Exemple :

```
import java.util.* ; //Import ici, avant le commentaire javadoc
/**
 * Une classe énumérant ses arguments.
 * Par exemple:
 * <pre>
 *    java Bonjour2 un
 *    bonjour
 *    0 : un
 * </pre>
 *
 * @author  Marc Martin
 * @version %I%, %G%
 */
public class Bonjour2
...
```

Génération : Pour générer la page Bonjour2.html, faire :

> javadoc Bonjour2.java

Exemple de documentation HTML

Class	Tree	Deprecated	Index	Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class Bonjour2

java.lang.Object | +--
Bonjour2

public class **Bonjour2**
extends java.lang.Object
Une classe énumérant ses arguments.
Par exemple:
java Bonjour2 un bonjour 0 : un

Constructor Summary	
Bonjour2()	

Method Summary

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

Methods inherited from class java.lang.Object

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

Constructor Detail

Bonjour2

public **Bonjour2**()

Method Detail

main

public static void **main**(java.lang.String[] arguments)

Exécution 1/2

- Qui exécute le programme que vous avez tapé et compilé ?
  - Le processeur directement ? instruction par instruction ?
  - A quoi servirait le système d'exploitation alors ?
  - A quoi servirait la machine virtuelle alors ?
- Le système d'exploitation alloue un processus à la machine virtuelle
  - Qui partage le CPU avec les autres processeurs
  - Qui utilise un espace de mémoire virtuelle

Exécution 2/2

- La machine virtuelle JAVA alloue des Threads (fils d'exécution) aux programmes JAVA
  - Un pour l'initialisation
  - Un pour appeler les méthodes « callback » de l'interface (si votre programme est abonné !)
  - Un par Thread que vous créez vous-même
- Les Threads sont gérés par l'OS.
- La machine virtuelle termine lorsque le dernier thread du dernier programme se termine.
  - Tout l'espace mémoire est alors libéré (y compris la JVM)
- Ou bien appel direct à System.exit(0);

UNIVERSITÉ PARIS-SUD 11

Plan 2 : Structures fondamentales

1. Types et Variables
2. Affectations
3. Conversions
4. Constantes
5. Conversions des types primitifs
6. Ecritures d'entiers et de réels
7. Types de données primitifs
8. Opérateurs arithmétiques
9. Opérateurs relationnels et booléens
10. Tests
11. Boucles
12. Contrôles 1/2
13. Contrôles 2/2
14. Mots Réservés
15. Caractères d'échappement Java
16. Chaînes 1/2
17. Chaînes 2/2
18. Chaînes (entrée)
19. Chaînes (sortie)
20. Tableaux 1/2
21. Tableaux 2/2

Types et variables

- Non objet : boolean, char, byte, short, int, long, float, double
- Version objet des mêmes concepts (Boolean avec majuscule, Float, etc.)
- Classe java : mot clef class
- Déclaration de variable : Modificateur NomType nomVariable (=valeur);
- La portée de la variable est celle du bloc

```
public class ProgVar {
  public static void main (String[] args) {
    int varGlob = 0;
    {
      int varLoc = 0;
      varLoc     = varLoc +1;
    }
    // erreur
    varGlob = varLoc+1;
}
```

Attention à la syntaxe JAVA

- Un bloc n'est pas obligatoirement un if, un for ou un while
- VarLoc++; marche aussi
- l'initialisation des variables n'est pas obligatoire mais préférable

Affectations

- Comme en C, opérateur =
- Une variable ne peut pas être utilisée avant d'être initialisée (Float Temperature = null ;)
- Interdit de poser une affectation comme condition
  - ~~if (maVar = getVar()) { … }~~
- L'affectation de deux variables objets est une *coréférence*
- o1 = o2; // o1 et o2 désigne le même objet
- Certaines valeurs spéciales sont des constantes

```
float largestFloat = Float.MAX_VALUE;
```

Conversions

- La conversion numérique est faite automatiquement vers le type le plus riche dans une opération arithmétique
- La conversion peut être faite explicitement vers un type plus pauvre

```
double natMoy = 2.1;
int petiteFamilleType;
```

Attention à la syntaxe JAVA

- petiteFamilleType est initialisée après sa déclaration mais avant d'être utilisée
- 2.7 est un double, 2.7f est un float

```
petiteFamilleType = (int)natMoy ;  // vaut 2
Int Alloc = petiteFamilleType * 1500;
```

Hiérarchie des types : byte < short < int < long < float < double

Constantes

- Déclarées avec le mot-clé `final`.

```
final int HEURES_DANS_JOUR = 24;
int jours;
int heuresDansJours;
jours = 7;
heuresDansJours = jours * HEURES_DANS_JOUR;
```

Attention à la syntaxe JAVA

- Les constantes sont en majuscules avec des _ entre les mots
- Les Variables commencent par une minuscule et utilisent uns majuscule à chaque nouveau début de mot

Conversions des types primitifs

Conversion de	*Conversion vers* boolean	byte	short	char	int	long	float	double
boolean	-	N	N	N	N	N	N	N
byte	N	-	Y	C	Y	Y	Y	Y
short	N	C	-	C	Y	Y	Y	Y
char	N	C	C	-	Y	Y	Y	Y
int	N	C	C	C	-	Y	Y*	Y
long	N	C	C	C	C	-	Y*	Y*
float	N	C	C	C	C	C	-	Y
double	N	C	C	C	C	C	C	-

Y = YES

N = No

C = Cast (besoin de conversion explicite)

Ecritures d'entiers et de réels

- 255 == 0xff == 0377
- 1024 == 1024L == 0x400L
- 0.01 == .01 == 1e-2d == 1e-2f
- double inf = 1/0;
- double neginf = -1/0;
- double neg0 = -1/inf;
- double NaN = 0/0; (Nan = not a number)

Types de données primitifs

Mot clé	Description	Taille et Format
(entiers)		
byte	Byte-length integer	8-bit two's complement
short	Short integer	16-bit two's complement
int	Integer	32-bit two's complement
long	Long integer	64-bit two's complement
(nombres réels)		
float	Single-precision floating point	32-bit IEEE 754
double	Double-precision floating point	64-bit IEEE 754
(autres types)		
char	A single character	16-bit Unicode character
boolean	A boolean value (true or false)	true or false

Opérateurs arithmétiques

- Opérateurs numériques
  - + - * / et += -= *= /=
- Division entière
  - / %
- Exposant
  - y = Math.pow(x, a);
- Incrément et décrément :
  - i++;
  - x = y + --j;

Méthodes de la classe java.lang.Math

```
public static final double E;
public static final double PI;
public static double abs(double);
public static float abs(float);
public static int abs(int);
public static long abs(long);
public static native double acos(double);
public static native double asin(double);
public static native double atan(double);
public static native double atan2(double,
double);
public static native double ceil(double);
public static native double cos(double);
public static native double exp(double);
public static native double floor(double);
public static native double log(double);
public static double max(double, double);
public static float max(float, float);
```

Méthodes de java.lang.Math

```
public static int                    max(int, int);
public static long                   max(long, long);
public static double                 min(double, double);
public static float                  min(float, float);
public static int                    min(int, int);
public static long                   min(long, long);
public static native double          pow(double, double);
public static synchronized double    random();
public static native double          rint(double);
public static long                   round(double);
public static int                    round(float);
public static native double          sin(double);
public static native double          sqrt(double);
public static native double          tan(double);
public static double                 toDegrees(double);
public static double                 toRadians(double);
```

Opérateurs relationnels et booléens

- **Opérateurs relationnels**
  Égalité/inégalité de deux opérandes : == !=
- **Opérateurs booléens**
  Opérateurs logiques : &&(and) ||(or) !(not)
  Opérateurs binaires : &(bitwise and) |(bitwise or) ^(xor) ~(not)
- **Préséance** des opérateurs entre eux:
  [] () .(méthode)
  ! ~ ++ --
  * / %
  + -
  << >>
  < >
  = >= instanceof
  == !=
  &
  ^
  |
  &&
  ||
  = += -= *= /= %= &= |= ^=

Tests

- Comme en C

```
if (condition) {...}
else {...};

switch nomVariable
{
    case valeur1 : {...
        break;
    }
    ...
    case valeurn : {...
        break;
    }
    default : {...
        break;
    }
} ;
```

Attention à la syntaxe JAVA

- condition = QUE BOOLÉEN
- nomVariable = QUE int
- break; OBLIGATOIRE !

Boucles

- Comme en C

```
for (NomType compteur=valDebut; (condition);(expression))
     {...};
For (int i=0; i<20; i++) {}

while (condition)
     {...};
While (i!=20) {i++;}

do
...
while (condition);
Do {i++;} while (i!=20)
```

Contrôle 1/2

- Interruption de boucle
- Interruption non étiquetée : sortie de la boucle la plus haute.

```
while( i <= 100 ) {
    i += 10;
    if( i >= 2 * n + 1 ) {
        break;
    }
}
```

- Interruption étiquetée : sortie d'une boucle imbriquée.

```
boucle_a_interrompre:
while( i <= 100 ) {
    i += 10;
    while( i < j ) {
        i++;
        if( i >= 2 * n + 1 ) {
            break boucle_a_interrompre;
        }
    }
}
```

Contrôle 2/2

- Continuation de boucle = court-circuit de la fin d'une itération

```
var x = 0;
while (x >= 10)
{
  x++;
  if (x == 5)
  {
    continue;
  }
  document.write(i + '\n');
}
```

Produira

```
> 1 2 3 4 6 7 8 9 10
```

Mots réservés

abstract	double	int	strictfp **
boolean	else	interface	super
break	extends	long	switch
byte	final	native	synchronized
case	finally	new	this
catch	float	package	throw
char	for	private	throws
class	goto *	protected	transient
const *	if	public	try
continue	implements	return	void
default	import	short	volatile
do	instanceof	static	while

Caractères d'échappement Java

Caractère	Valeur Unicode	Description
\b	\u0008	effacement en arrière *BS* (backspace)
\t	\u0009	tabulation horizontale *HT* (horizontal tab)
\n	\u000a	fin de ligne *LF* (line feed)
\f	\u000c	saut de page *FF* (form feed)
\r	\u000d	retour chariot *CR* (carriage return)
\"	\u0022	guillemet double "
\'	\u0027	guillemet simple '
\\	\u005c	anti-slash \
\XXX	\000 - \377	Caractère Latin-1 en base 8
\uXXXX	\u0000 - \u00ff	Caractère unicode

Chaînes 1/2

- Affectation et concaténation

```
String intro = "Ce programme peut écrire ";
String affiche;

affiche = intro + "une chaîne de " + (50 + 2) + "caractères.";
System.out.println( affiche );
```

- Sous-chaînes

```
String mot_construit = "Sous-chaîne";
String racine = mot_construit.substring( 5 ,  mot_construit.length()
   - 5 );
char tiret = mot_construit.charAt( 4 );
```

- Égalité entre chaînes
  - Égalité de valeurs : méthode equals()
  - Identité (à ne pas utiliser) : ==

```
if( racine.equals( "chaîne" ) ) {
     System.out.println( "Égalité de valeurs" );
}
```

Chaînes 2/2

- Mutabilité
  - Les String ne sont pas mutables. Le contenu ne peut pas être modifié. Toute opération créé une nouvelle chaîne.

```
String text = "Le vilain petit canard";

text += " devient un beau cygne !";
// provoque la suppression de "Le vilain petit canard«
```

- StringBuffer
  - Sont mutables (utilisés pour les +)

```
StringBuffer ch = new StringBuffer("Jean fait comme ");
ch = ch.append("il peut pour travailler\n");
ch = ch.replace(ch.indexOf("il ")
                                    + " ".length(),
    ch.indexOf("pour"), "veut ");
System.out.println (ch);
```

Chaînes : Entrée

- Lecture d'une chaîne au clavier

```
InputStreamReader is = new InputStreamReader( System.in );
BufferedReader bf    = new BufferedReader( is );

String chaineAAnalyser = "";
System.out.print( "Entrez une chaîne : " );
try{
    chaineAAnalyser = bf.readLine();
}
catch(IOException IOException_Arg) {
    System.out.println(IOException_Arg.getMessage()) ;
}
```

Chaînes : Affichage

- **Sortie non formatée**
  - Affichage sans limite sur le nombre de décimales

```
double x = 1.0 / 3.0;
System.out.println( x );
```

Formattage des entiers

- Création d'un format d'affichage numérique

```
DecimalFormat df = new DecimalFormat( "0.###" );
System.out.println( df.Format( x ) );
```

Formats :

0	chiffre
#	chiffre non visible si 0 de complément
.	symbole décimal
,	séparateur de groupements
;	séparateur du format nombres positifs et du format nombres négatifs
-	préfixe des nombres négatifs
%	affichage d'un pourcentage

Tableaux 1/2

- Principes et déclarations
  - Les tableaux sont des collections de taille fixe d'objets de même type.

```
double[] x = new double[10];
```

- Allocation et initialisation simultanées
  - L'appel de new est alors inutile.

```
double[] y = { 0.0, 0.25, 0.5, 0.75, 1.0};
```

Tableaux 2/2

- Affectation de tableaux
  - Copie de valeur

```
y[ 3 ] = 0.3;
for (int i=0; i<y.length; i++) {y[i]=0.5;}
```

  - Copie de pointeur

```
double[] z = y;
```

- Tableaux multi-dimensionnels
  - Il s'agit d'un tableau de tableaux pour lequel chaque rangée est accessible individuellement.

```
double[][] matrice_carree;
matrice_carree = new double[3][3];
matrice_carree[2] = y;
```

Plan 3 : Objets et Classes

1. Introduction
2. Exemple
3. Notion d'encapsulation
4. Concepts Objets (1)
5. Concepts Objets (2)
6. Concepts Objets (3)
7. Notions de POO
8. Définition de classe
9. Exemple Classe Point
10. Constructeurs
11. Exemple d'instance (1)
12. Exemple d'instance (2)
13. Surcharge
14. Référence this
15. Conseils de conception (1)
16. Conseils de conception (2)
17. Conseils de conception (3)

Introduction

- Programmation fonctionnelle
  - Des fonctions
- Programmation impérative ou procédurale
  - Des ordres (éventuellement avec des fonctions)
- Programmation Objet
  - Un programme divisé en plusieurs sous-modules indépendants
  - Les modules sont chargés en mémoire à l'exécution
  - Le concepteur fait évoluer ses modules en parallèle
- Les modules sont typés comme des int et des float
  - Le type de module est aussi important que le module lui-même

Exemple

```
public class etudiant {
    String nom;
    int promo;
    String traiterExam(String enonce)
    {…}
}

public class prof {
    String nom;
    String faireSujetExam()
    {…}
    int corrigerExam(String copie)
    {…}
}
```

Notion d'encapsulation

- Définition
  - Combinaison de données et de méthodes dans un emballage unique (un objet)
  - L'implémentation n'est pas accessible aux utilisateurs
  - L'altération ou l'accès à l'état d'un objet se fait essentiellement par ses méthodes
  - Encapsulation = regroupement et organisation de l'information.

Concepts Objet 1/3

- Classe
  - Un patron
  - Une description de la boîte noire
  - Un type
- Instance
  - Un modèle réalisé avec le patron
  - Une boîte noire dynamique
  - Un **objet**
- Référence
  - Un pointeur vers une instance
  - Une adresse d'une boite noire dynamique
  - Un fournisseur de service

Homme

Bill Clinton
Georges W. Bush
Hillary Clinton

Président des USA
Conjoint du
Président des USA

Concepts Objet 2/3

- En mémoire
  - La classe est chargée une seule fois
  - Les instances sont chargées dynamiquement autant de fois qu'il y a d'instances dans le programme
  - Un compteur est créé pour chaque instance
  - A chaque fois qu'une nouvelle variable fait référence à l'instance compteur++
  - A chaque fois qu'une référence est perdue (sortie de boucle) compteur- -
  - Quand compteur==0 => JAVA libére la mémoire
  - Quand JAVA libère la mémoire d'un objet => compteur - - pour les attributs de l'instance détruite!!!
    - Si dernière référence => destruction en chaîne
    - Si co-référence : problème (Simple OK pour java)

Concepts Objet 3/3

- Méthode
  - Une fonctionnalité
    - Interne
    - Externe
- Variable d'instance
  - Variable
  - Appartient à un objet
- Variables de classe
  - Partagé par toutes les instances de la classe
- Constantes (d'instance ou de classe c'est pareil !!!)
- Programmation Objet = un ensemble d'objets
- Objet = données + méthodes

Attention à la terminologie

• Classe / Instance c'est clair

• Objet = moins clair, pourtant c'est le terme le plus utilisé

Notions de Programmation OO

- Responsabilités
  - Chaque objet est responsable de l'accomplissement des tâches qui lui sont associées.
  - Les objets clients envoient des messages aux objets serveurs.
- Eléments
  - Les éléments de base sont des objets qui sont des instances d'une classe :
  - les propriétés génériques d'un objet sont au niveau de la classe,
  - l'état d'un objet est décrit au niveau de l‘instance,
  - les classes sont organisées hiérarchiquement et bénéficient d'un mécanisme d'héritage automatique.

Définition de classe

- Les variables d'instance définissent l'état d'une instance.

```
public class etudiant {
    String nom;
    int promo;
}
```

- Méthodes de construction
  - Constructeur : Exécuté à la création d'une nouvelle instance tout de suite après le chargement en mémoire.
- Méthodes d'altération
  - initialisent, modifient ou annulent la valeur des variables d'instance.
- Méthodes d'accès
  - lisent les valeurs des variables d'instance.

Exemple : Classe Point

```
public class Point                          // une classe
{
  int x;                                    // variable d'instance
  int y;                                    // variable d'instance

  public int getX() {                       // accès
    return x;
  }

  public void setX(int abscisse) {          // altération
    x = abscisse;
  }

  public int getY() {
    return y;
  }

  public void setY(int ordonnee) {
    y = ordonnee;
  }
}
```

Attention aux conventions Java

- setXXX() et getXXX() sont plus ou moins officiels et aident beaucoup à la lisibilité
- Regroupement du set et du get correspondant plutôt que tous les get (resp. set) ensemble

Constructeurs

- Des méthodes permettant de préparer l'objet à utiliser
  - initialiser les variables d'instance
  - Toute classe possède un constructeur par défaut qui initialise les variables à leurs valeurs par défaut :

```
NomClasse()
```

  - Si une classe a un constructeur => plus de constructeur par défaut
  - Il **peut** y avoir plus d'un constructeur par classe
- L'opérateur **new** permet de
  - Charger les méthodes de la classe en mémoire si c'est la première fois qu'un objet de ce type est invoqué
  - Créer l'instance en mémoire (Réserve exactement la taille de mémoire pour les données et les initialisent avec leur valeurs par défaut)
  - Lancer le constructeur (Correspondant aux arguments donnés)
  - Le constructeur peut appeler d'autres méthodes de l'instance (y compris d'autres constructeurs) ou de la classe.

Exemple d'instance : unPoint

```
import Point;              // on a besoin de la classe Point

public class TestPoint {
  public static void main(String[] arg) {
    Point unPoint;         // une variable objet
    unPoint=new Point();   // création d'une instance
    unPoint.setX(10);
    unPoint.setY(100);
    System.out.println("point\n\t d'abscisse " +
                       unPoint.getX() +
                       "\n"+"\t et d'ordonnée "+
                       unPoint.getY());
  }
}// Fin class TestPoint
```

- Version abrégée : `Point unPoint = new Point();`

Exemple : Homme/President

```
public class USA {
  protected    Homme        president    = null;
  private      Population[] etats        = new Population[NB_ETATS];
  public final int          NB_ETATS     = 50;

  public static void main(String[] arg) {
    Homme BillClinton    = new Homme("Bill", "Clinton", ...);
    Femme HillaryClinton = new Femme("Hillary", "Clinton", ...);
    BillClinton.setConjoint(HillaryClinton);
    HillaryClinton.setConjoint(BillClinton);
    setPresident(BillClinton);
  }

  // ne doit etre appelle que par election()
  private setPresident(Homme NewPresident_Arg) {
    president = NewPresident_Arg
  }

  // ne doit etre appelle que par l'objet AssembleeNationale
  public election(Homme Candidat1_Arg, Homme Candidat1_Arg) {
    for (int i=0; i<NB_ETATS; i++) {
      Homme Choix = etats[i].choosePresident(Candidat1_Arg, Candidat2_Arg);
      ...
      setPresident(Resultat);
    }
  }
}// Fin class USA
```

Questions

  - BillClinton n'est pas une instance (référence)
  - President n'est pas une instance (classe)
- Comment appeler l'instance ?
- Comment être sûr qu'une référence pointe vers une instance ?
- Et si le président est une femme ?
- Et si il y a plus de deux candidats ?
- Qui créé AssembleeNationale ?
- Si on a une classe Assemblee et plusieurs instances (France, GB, USA, …) comment être sur que l'assemblée américaine est la seule à convoquer une élection aux USA ?
- Qui est au dessus de qui ?

Notion de surcharge

- Plusieurs méthodes de même nom mais de signatures différentes (liste d'arguments+type de retour).

```
public void setEtudiant(String unNom, int unePromo) {
    nom   = unNom;
    promo = unePromo;
}
public void setEtudiant(String unNom) {
    nom   = unNom;
    promo = 0;
}
```

- La surcharge permet une initialisation explicite des variables d'instance non garnies par un constructeur ou une méthode d'altération.
- Un constructeur par défaut est un constructeur sans argument.

```
public Etudiant() {
    nom = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
    promo = 0;
}
```

Auto-Référence (this)

- Dans une méthode, le mot-clé this fait référence à l'objet sur lequel opère la méthode.

```
System.out.println( this );
```

- On peut imprimer toute instance en JAVA. L'impression fait appel à la méthode toString() (héritée de Object).
- Dans un constructeur, le mot-clé this en début d'un constructeur désigne un autre constructeur de la même classe.

```
public Etudiant() {
  this( "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX" );
}
```

- À consommer avec modération !

Conseils de conception 1/3

- Quelques règles à suivre
  - Privatiser les données : principe d'encapsulation
  - Initialiser les données : éviter de faire appel aux initialisations par défaut qui sont source d'erreur
- Limiter les variables d'instances : les regrouper dans une sous-classe si c'est nécessaire
- Utiliser un format standard pour la définition des classes : (1) constructeurs, (2) éléments publics, (3) éléments accessibles au paquetage, (4) éléments privés
- Utiliser un outil (IDE) qui tienne compte de l'organisation Package/Classe/Fonction
- Utiliser un navigateur pour l'aide et pour google !

Conseils de conception 2/3

- Nommer explicitement : classes et variables doivent avoir des noms clairs (donc longs) et explicitant leur type (verbes pour méthodes et noms pour classes).
- Diviser pour mieux régner : les classes trop lourdes doivent être disperser en plusieurs classes
  - Commencer par une analyse du problème
  - Imaginer une solution composée de travailleurs humains
  - Traduire en objets en tenant compte des spécificités des ordinateurs par rapport aux humain
    - Temps de calcul, place en mémoire
    - Cohérence et nombre des méthodes
    - Intuitivité de votre organisation du travail par rapport aux autres

UNIVERSITÉ PARIS-SUD 11

Conseils de conception 3/3

- Faire des schémas
  - Représentant les classes
    - Pour comprendre les interactions entre les classes (qui a besoin d'une référence sur qui)
    - Pour savoir ou placer une fonction
  - Représentant les appels de fonction
    - Pour savoir quels objets sont traversés
    - Pour savoir où récupérer les données à mettre en argument
- Les fonctions qui ne font rien d'autre qu'appeler une autre fonction sont fréquentes
  - Dans les langages à objet compilés on les « inline » pour ne pas perdre de temps à l'exécution

Exercice : la classe disque

- Créer la classe Disque
  - Diamètre
  - Epaisseur
  - Matière (bois, fer ou verre)
  - Peint ou brut
- Méthodes de Calcul du volume ($\pi \times R^2$), de la masse (vol*masse vol fer=7,8 kg/l bois=0.8, verre=2.5)
- Créer la classe Pile de Disque
  - Son ensemble d'éléments
  - Des fonctions pour se faire manipuler de l'extérieur
    - empiler()
    - depiler()
    - sommet()
    - enlever() = depiler()+sommet()
- Créer la classe InterfaceUtilisateurTexte
  - Trois Piles (A, B, C),