IIS

INFORMATIQUE I et II
2001 - 2002

Résumé du langage

François Grize

UNIVERSITE DE LAUSANNE

TABLES DES MATIERES

ANNEXE 1 : Principaux composants graphiques

ANNEXE 2 : La classe String

ANNEXE 3 : Quelques caractères spéciaux

ANNEXE 4 : Hiérarchie partielle des classes dans AWT

ANNEXE 5 : Quelques classes en Java

LIVRES DE REFERENCE

XI PROGRAMMATION ORIENTEE OBJETS

XI.1 Définitions de base

Classe

Concept qui permet d'*encapsuler* des informations et des comportements communs à un ensemble d'objets. Fabrique d'objets.

Objet

A partir d'une classe, on peut fabriquer autant d'objets qu'on veut. On parle aussi d'**instance** de classe.

Attribut

Ce sont les informations de la classe.

Méthode

Ce sont les comportements de la classe.

Message

Pour activer une méthode, on envoie un message à un objet.

Exemple I.1-1 : Déclaration d'une classe

```
class Personne {
    String prenom;
    String nom;
    // méthode
    String presenteToi( ) {
       return prenom + ″ ″ + nom;
    }
}

Personne moi;
// instanciation de l'objet moi
moi = new Personne();
moi.prenom = ″Jules″;
moi.nom = ″Dupont″;
```

Remarques:

1) Une méthode comporte toujours deux parties :

- une *entête* (`String presenteToi( )`), qui précise son mode d'emploi
- un *corps* (`{...}`), qui contient les instructions à exécuter;

2) L'opérateur `+` permet la concaténation de chaînes;

3) L'opérateur **`new`** permet d'instancier une classe, c'est-à-dire de créer un objet;

4) `moi.presenteToi()` retourne la chaîne `"Jules Dupont"`.

XII SYNTAXE

XII.1 Vocabulaire

a) Les mots réservés

Les mots suivants sont réservés à une utilisation bien précise dans le langage Java. Ils ne peuvent pas être utilisés comme noms de classes, de méthodes ou de variables :

Tableau 1:

abstract	abstrait	voir IX.2
boolean	booléen	voir IV; VI; VII
break	rupture	voir VI.3; VII.4
byte	octet	voir III.1
case	au cas où	voir VI.3
catch	capturer	
char	caractère	voir II.3.c
class	classe	voir I.1; II.2; II.3; III.1; VIII; IX.2
const	constante	non utilisé
continue	continuer	
default	défaut	voir VI.3
do	faire	voir VII.2; VII.4
double	double	voir II.3.b; III.1
else	sinon	voir VI.2
extends	étendre	voir VIII.5
false	faux	voir IV
final	final	voir VIII.7; IX.2
finally	finalement	
float	flottant	voir II.3.b; III.1
for	pour	voir VII.3; VII.4
goto	aller à	non utilisé
if	si	voir VI
implements	implémente	
import	importer	voir IX.1
instanceof	instance de	
int	entier	voir III.1

Tableau 1:

interface	interface	voir II.2
long	long	voir II.3; III.1
native	natif	
new	nouveau	voir I.1; X.1
null	nul	voir VIII.2
package	paquetage	voir IX
private	privé	voir VIII.6
protected	protégé	voir VIII.6
public	public	voir VIII.6; IX.2
return	retourner	voir I.1; VIII.3
short	court	voir III.1
static	statique	voir VIII.7
super	super	voir VIII.5
switch	aiguillage	voir VI.3; VII.4
synchronized	synchronisé	
this	celui-ci	voir VIII.2
throw	lever (une exception)	
throws	lève	
transcient		
true	vrai	voir IV
try	essayer	
void	vide	voir VIII.3
volatile	volatile	
while	tant que	voir VII.1; VII.3; VII.4

b) Quelques «symboles de ponctuation»

1) séparateurs: ( ) {} [] ; , .
2) opérateurs: + - * / ...
= < > ...
&& || += -= ...

XII.2 Grammaire

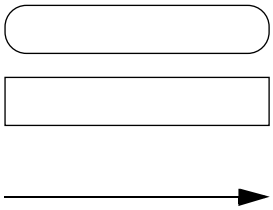
a) Schémas syntaxiques

Les éléments de vocabulaire sont entre guillemets.
{...} : 0 ou plusieurs fois
[ ] : optionnel
| : choix

Exemple II.2-1 : Schéma syntaxique du `type_declaration`

```
type_declaration
[doc_comment] {class_declaration|interface_declaration} ";"
```

b) Diagrammes syntaxiques

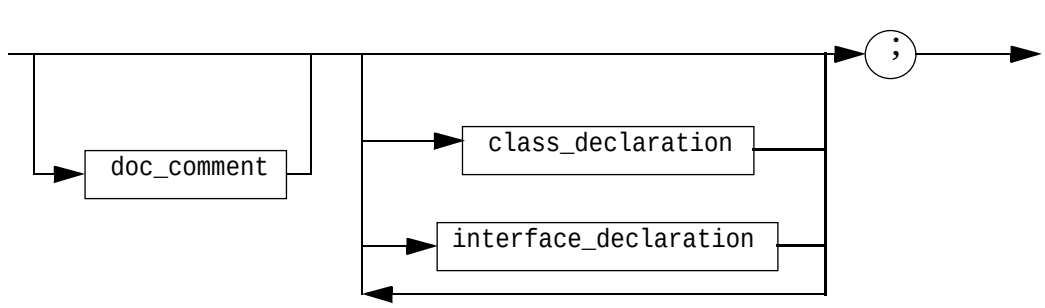


un élément du vocabulaire

une entrée vers une règle de grammaire

comment suivre la règle

Exemple II.2-2 : Diagramme syntaxique du `type_declaration`



On trouve tous les schémas syntaxiques et les diagrammes syntaxiques de Java à :
http://cui.unige.ch/java/JAVAF/compilation_unit.html

XII.3 Unités syntaxiques

a) Identificateurs

Une suite de lettres et de chiffres ainsi que les caractères $ (dollar) et _ (souligné). Le premier caractère doit être une lettre.

Contrairement à certains langages, on distingue les majuscules des minuscules.

Par convention, les identificateurs de classe commencent par une majuscule, les autres par une minuscule.

Pour séparer les mots, on utilise une majuscule.

Exemple II.3-1 : Les identificateurs

```
ceciEstUnIdentificateur
String
boeing707
MAX_VALUE
çàEtLà
àCôté
αγετη
```

b) Littéraux numériques

- Les *entiers* sont constitués d'une suite de chiffres, éventuellement précédée par une marque de base (`0x` ou `0X` pour héxadécimal et `0` pour octal), éventuellement suivie du suffixe `L` ou `l` (pour le type **`long`**).

Exemple II.3-2 : Les entiers

```
2514
0xDadaCafe
1998
0777L
21474864l
```

- Les *flottants* sont constitués des parties suivantes:
  - une partie entière
  - un point décimal
  - une partie fractionnaire
  - éventuellement un exposant
  - éventuellement les suffixes `F` ou `f` (pour le type **`float`**) et `D` ou `d` (pour le type **`double`**).

    Si le suffixe est absent, le nombre est de type **`double`**.

  L'exposant est indiqué par la lettre `E` ou `e` suivi d'un entier avec ou sans signe.

Exemple II.3-3 : Les flottants

```
3.14159
2.0f
6.02213e+23
1.0E-9d
1e10
```

c) Littéraux caractères et chaînes

- Un littéral caractère. C'est un caractère (éventuellement une séquence *escape*) entouré d'apostrophe: `'a'  'Ω'.`
- Un littéral chaîne. C'est zéro ou plusieurs caractères entourés de guillemets:

  `" ", "ceci est une chaîne", "" (chaîne vide).`

d) Commentaires

- Une ligne de commentaire. Le texte qui commence par `//` est ignoré jusqu'à la fin de la ligne.
- Un texte qui commence par `/*` et qui se termine par `*/.`

XIII. TYPES NUMERIQUES ET EXPRESSIONS ARITHMETIQUES

XIII.1 Types numériques

1) Les types entiers sont:

- **`byte`** : `-128` à `127`
- **`short`**: `-32768` à `32767`
- **`int`** : `-2147483648` à `2147483647`
- **`long`** : environ 10^{-18} à 10^{18}

2) Les types réels sont:

- **`float`** : représentés sur 32 bits
- **`double`** : représentés sur 64 bits.

3) Conversion de types:

(`nom de type`) `expression`

convertit *`expression`* dans le type précisé par *`nom de type`*.

Exemple: `(`**`int`**`)` `4.7` `retourne` l'entier `4`

Les constantes (littéraux numériques) de ces types sont définies en II.3.b

a) Opérateurs unaires

+	identité
-	inversion de signe
++	incrémentation préfixée et postfixée
--	décrémentation préfixée et postfixée

Exemple III.1-1 : Opérateurs unaires

```
++v  est équivalent à v=v+1
--v  est équivalent à v=v-1
v++  retourne v puis l'incrémente
v--  retourne v puis le décrémente
```

b) Opérateurs binaires

+	addition
-	soustraction
*	multiplication
/	division
%	reste de la division entière; opérateur *modulo*

Remarques:

1) Si le membre de gauche et le membre de droite de l'opérateur `/` sont tous deux de type entier, c'est la division entière qui s'applique.

   Exemple: `7 / 2` retourne `3;`

2) Lorsqu'il y a mélange d'opérandes entiers et réels, les opérations se font en réel et le résultat est réel;

3) Les multiplications et les divisions sont traitées avant les additions et les soustractions. En cas d'égalité de priorité, l'évaluation se fait de gauche à droite. Les parenthèses permettent de changer de priorités;

4) La classe `java.lang.Math` contient les principales fonctions arithmétiques standard.

XIV CONDITIONS ET EXPRESSIONS BOOLEENNES

Une expression booléenne (en référence au mathématicien britannique George Boole) peut prendre soit la valeur faux (false) soit la valeur vrai (true).

XIV.1 Type boolean (booléen)

Les variables booléennes ne peuvent prendre que deux valeurs: faux (`false`) ou vrai (`true`).

XIV.2 Tables de vérité

a) Opérateurs unaires

Soit p une variable de proposition qui prend soit la valeur 0 (en Java: `false`), si la proposition est fausse, soit la valeur 1 (en Java: `true`), si la proposition est vraie.
On a alors $2^2 = 4$ opérateurs unaires:

- identité (laisse tout comme c'est)
- transforme tout en 1
- transforme tout en 0
- négation notée ¬ ou ~ (non; en Java !)

p	¬p
1	0
0	1

b) Opérateurs binaires

Soit `p` et `q` deux variables de proposition qui prennent soit la valeur 0 (en Java: `false`), si la proposition est fausse, soit la valeur 1 (en Java: `true`), si la proposition est vraie.
On a alors $2^4 = 16$ opérateurs binaires:

- **conjonction** notée ∧ (et; en Java **&**).

p	∧	q
1	1	1
1	0	0
0	0	1
0	0	0

- **disjonction** notée V (ou; en Java **|**).

p	V	q
1	1	1
1	1	0
0	1	1
0	0	0

- **disjonction exclusive** notée w (ou exclusif; en Java ∧).

p	w	q
1	0	1
1	1	0
0	1	1
0	0	0

Exemple IV.2-1 : Evaluation d'expression booléenne

Si p = false, q = true, r = true

(p V q) Λ r vaut true

c) Lois de Morgan

1) ¬ (p Λ q) est équivalent à ¬ p V ¬ q
2) ¬ (p V q) est équivalent à ¬ p Λ ¬ q

- **Démonstration 1**:

¬	(p	Λ	q)
0	1	1	1
1	1	0	0
1	0	0	1
1	0	0	0

¬p	V	¬q
0	0	0
0	1	1
1	1	0
1	1	1

- **Démonstration 2**: exercice !

XIV.3 Opérateurs booléens

Tableau 2:

Opération	**Opérateur Java**
négation (¬)	!
conjonction (Λ)	&
disjonction (V)	|
disjonction exclusive (W)	^
conjonction conditionnelle	&&
disjonction conditionnelle	||

Remarques:

- p && q: q n'est pas évalué si p est faux;
- p || q: q n'est pas évalué si p est vrai.

XIV.4 Opérateurs de comparaison

Tableau 3:

Opération	Signification	Opérateur Java
égalité	=	==
différence	≠	!=
inférieur à	<	<
inférieur ou égal à	≤	<=
supérieur à	>	>
supérieur ou égal à	≥	>=

XIV.5 Opérateur conditionnel

condition ? *expression-1* : *expression-2*

retourne *expression-1* si *condition* est vrai
sinon retourne *expression-2*

Exemple IV.5-1 : Opérateur conditionnel

```
int n;
String s;
...
S = ″soit ″ + n + ″ franc″ + (n > 1 ? ″s″ : ″″);
```

XV INSTRUCTIONS D'AFFECTATION ET INSTRUCTION COMPOSEE

XV.1 Instructions d'affectation

Deux formes possibles :

1) *variable* = *expression*;

affecte *expression* à *variable*.

2) *variable* *opérateur binaire* = *expression*;

est équivalent à:

variable = *variable* *opérateur binaire* *expression*;

Remarque:

opérateur binaire : un des cinq opérateurs binaires arithmétiques (cf. III.1.b).

XV.2 Instruction composée et bloc

a) Instruction composée

Permet de regrouper plusieurs instructions entre { et }, là où la syntaxe impose une seule instruction.

b) Bloc

Une instruction composée qui peut comprendre des déclarations de variables locales.

XVI INSTRUCTIONS CONDITIONNELLES

XVI.1 Instruction conditionnelle simple

if (*condition*) *instruction*;

condition est une expression booléenne (cf. IV).
Si *condition* est vraie alors *instruction* est exécutée.

XVI.2 Instruction conditionnelle avec alternative

if (*condition*) *instructionSiVraie* **else** *instructionSiFaux*;

condition est une expression booléenne (cf. IV)
Si *condition* est vraie alors *instructionSiVraie* est exécutée
sinon *instructionSiFaux* est exécutée.

Remarque:

Lorsqu'on a des instructions conditionnelles imbriquées, le **else** se rapporte toujours au dernier **if**.

Exemple VI.2-1 : If - else

```
i=0; j=3;
  if (i==0)
       if (j==2)  i=4;
  else i=j;
```

i vaut 3. Afin d'éviter la confusion, le **else** devrait être aligné sous le dernier **if**!

XVI.3 Aiguillage

```
switch
(expression) {
        case constante-1 : suiteInstructions-1;
        case constante-2 : suiteInstructions-2;
        ...
        default : suiteInstructions;
 };
```

Remarques:

1) *expression* doit être de type caractère ou entier;

2) *constante-n* est une expression constante; elle peut contenir des opérateurs, mais doit pouvoir être évaluée à la compilation;

3) les constantes doivent être mutuellement exclusives;

4) si *expression* ne correspond à aucun cas prévu, l'instruction **switch** est ignorée, sauf s'il y a une partie **default**;

5) la partie **default** est évidemment optionnelle.

Attention:

Après l'exécution d'un branchement, on exécute le branchement suivant, sauf si on rencontre l'instruction **break**.

Exemple VI.3-1 : Aiguillage

```
int i, m;
...
switch (i) {
      case 1 : m=1;
      case 2 : m=2;
      break;
      default : m=3;
}
```

si i vaut 0 alors m vaut 3
si i vaut 1 alors m vaut 2
si i vaut 2 alors m vaut 2

XVII INSTRUCTIONS DE REPETITION

XVII.1 Instruction while (tant que)

while (*condition*) *instruction*;

condition est une expression booléenne.
Tant que *condition* est vraie, *instruction* est répétée.

XVII.2 Instruction do (faire)

do *instruction* **while** (*condition*);

condition est une expression booléenne.
Exécute *instruction* tant que *condition* est vraie.

XVII.3 Instruction for (pour)

for (*initialisation*; *condition*; *miseAjour*) *instruction*;

initialisation est une instruction d'affectation.
condition est une expression booléenne.
miseAjour est une expression.

Cette instruction est équivalente à:

```
initialisation;
while (condition) {
    instruction;
    miseAjour;
}
```

Remarque:

Dans la partie *initialisation*, il est possible de déclarer la variable de contrôle.

Exemple VII.3-1 : Instruction `for`

```
int somme = 0;
...
for (int i = 1;  i <= 10;  i++)
      somme  += i;
```

calcule la somme des 10 premiers entiers

XVII.4 Instruction break (rupture)

L'instruction **break** peut apparaître dans une instruction **switch** (cf. VI.3) ou dans n'importe quelle instruction de répétition (**while**, **do**, **for**). Elle permet de sortir de la boucle dans laquelle elle se situe.

Exemple VII.4-1 : Instruction `break`

```
/* recherche le premier nombre premier
dans l'intervalle min .. max  */

int  min,  max;
int  i;        // pour parcourir l'intervalle
...
for (i = min;  i <= max;  i++)
    if (estPremier (i))  break;
```

XVIII CLASSE

class *SonNom* *{*

variables

constructeurs

méthodes

}

Remarques:

1) chacune des trois parties du corps de la classe peut être absente;

2) on fait la distinction entre variable ou méthodes d'instance et variable ou méthode de classe (cf. VIII.7)

XVIII.1 Variable

Une variable (ou attribut) permet de préciser un trait commun à tous les objets de la classe.

Exemple VIII.1-1 : Variable

```
class Personne {
    String prenom, nom;
    int age;
    ...
```

Remarques:

1) Toute personne a un prénom, un nom et un âge;

2) Dans ce cas, on éviterait, par contre, de mettre un attribut salaire, car toute personne n'est pas nécessairement salariée.

XVIII.2 Constructeur

Un constructeur permet d'initialiser les variables d'un objet lors de sa création (**new**).

Un constructeur a le même nom que la classe qui le contient.

Une classe peut contenir plusieurs constructeurs pourvu qu'ils se distinguent par le nombre ou le type de leurs paramètres (cf. VIII.4).

Quand une classe ne prévoit aucun constructeur, Java fournit un constructeur par défaut, sans paramètre, qui initialise les variables d'instances avec des valeurs par défaut conformément à leur type. A savoir:

- pour les entiers: `0`
- pour les réels: `0.0`
- pour les booléens: `false`
- pour les caractères: `' '`
- pour les références à des objets: **`null`**

Exemple VIII.2-1 : Constructeur

```
class Personne {
     String prenom;
     String nom;
     int age

     // constructeurs
     Personne (String p, String n) {
          prenom = p;
          nom = n;
          age = -1;   //-1 si l'âge est inconnu
     }
     Personne (String p, String n, int a) {
          prenom = p;
          nom = n;
          age = a;
     }
}
...

Personne elle = new Personne ("Julie","Durand");
Personne lui = new Personne ("Alain","Térieur",24);
```

Remarque:

Dans le deuxième constructeur (celui qui a l'âge en paramètre), on aimerait pouvoir utiliser le premier; on peut le faire grâce au mot réservé **this.**

Exemple VIII.2-2 : Utilisation de `this`

```
Personne (String p, String n, int a){
     this (p,n);
     age = a;
}
```

XVIII.3 Méthode

Une méthode permet d'opérer sur un objet.

type unNom (*paramètres*) *{*

instructions

}

type : retourné par la méthode. Si elle ne retourne rien, on utilise le mot réservé **`void`**.
La valeur de la méthode est retournée par le mot réservé **`return`**, qui a aussi pour effet de rendre le contrôle à l'objet appelant.

paramètres : peut être absent

Exemple VIII.3-1: Méthode

```
   // méthode de la classe Personne
   // cf. Exemples VIII.1-1 et VIII.2-1

   String presenteToi( ) {
        String s;
        s = "Je m'appelle "+prenom+" "+nom;
        if (age >= 0) return s+" et j'ai "+age+" ans";
        else return s;
   }

...

elle.presenteToi( ) retourne:
Je m'appelle Julie Durand

lui.presenteToi( ) retourne:
Je m'appelle Alain Térieur et j'ai 24 ans
```

XVIII.4 Signature

Il s'agit du nom (d'un constructeur ou d'une méthode) ainsi que du nombre et du type de ses paramètres formels.

Une classe ne peut pas déclarer deux constructeurs ou deux méthodes ayant la même signature.

Exemple VIII.4-1: Deux méthodes de même signature

```
final void move(int dx, int dy)
      ...

void move(int x, int y)
      ...
```

Par contre, il est possible, dans une classe, de donner le même nom à deux constructeurs différents ou à deux méthodes différentes, pourvu que leur signature soit différente.

Exemple VIII.4-2: Méthodes de même nom, signature différente

```
System.out.println (512);
System.out.println ("toto");
```

XVIII.5 Le concept d'héritage

Une classe est toujours définie à partir d'une autre classe: sa *superclasse*. On parle aussi de *classe mère* et de *classe fille* (ou *sous-classe*). Une classe qui est définie sans préciser sa superclasse est implicitement une classe fille de `Object`. L'ensemble de toutes les classes Java (définies de manière standard ou par le programmeur) forme une *arborescence* dont la racine unique est `Object`.

class *Fille* **extends** *Superclasse* {...}

La sous-classe *Fille* *hérite* automatiquement toutes les propriétés de *Superclasse* à laquelle on n'ajoutera que les propriétés qui lui sont particulières.

Une sous-classe peut donc utiliser toutes les variables, constructeurs ou méthodes de sa superclasse et ceci transitivement jusqu'à la racine.

Une sous-classe peut déclarer un identificateur qui a le même nom qu'un identificateur déjà déclaré dans sa

superclasse. Il n'y a pas d'ambiguïté possible, car on se réfère toujours au dernier identificateur déclaré.

Pour désigner explicitement un identificateur de la superclasse, en particulier un de ses constructeurs, on utilise le mot réservé **super**.

Exemple VIII.5-1 : Identificateurs synomymes

```
class M {
    int a;
    ...
classe F extends M {
    String a;
    ...
    a = "Toto";
    super.a = 12
```

Exemple VIII.5-2: Héritage

```
class Salarie extends Personne {
    double salaire;
    // constructeur
    Salarie (String p, String n, double s) {
        super (p,n);
        salaire = s;
    }
}
```

Remarques:

1) Le mot réservé **super**, s'il n'est pas suivi d'un identificateur, désigne un constructeur qui doit être recherché dans la classe mère.
2) Si une classe fille ne prévoit aucun constructeur ou si elle ne fait pas explicitement appel à un constructeur de la classe mère, Java appelle le constructeur par défaut de la classe mère, c'est-à-dire le constructeur sans

paramètre (ici: Personne( ), qui n'a pas été prévu dans l'exemple VIII.2-1).

XVIII.6 Contrôle d'accès

Les entités (variables, constructeurs, méthodes) déclarées dans une classe peuvent être:

- **`public`**

dans ce cas, elles sont accessibles partout; autant à l'intérieur qu'à l'extérieur de la classe;

- **`private`**

dans ce cas, elles ne sont accessibles qu'à l'intérieur de la classe;

- **`protected`**

dans ce cas, elles ne sont accessibles qu'à l'intérieur de la classe et de l'ensemble de ses sous-classes.

Remarques:

1) Par défaut, les entités sont déclarées comme **`public`**;
2) Par soucis de lisibilité, on utilisera quand même explicitement le mot réservé **`public`**;
3) Afin de ne pas devoir entrer dans les détails de l'implémentation, l'utilisateur d'une classe ne devrait pas pouvoir manipuler directement ses attributs si ce n'est à l'aide de méthodes prévues à cet effet (penser aux méthodes `get...( )` et `set...( )`).

XVIII.7 Variables et méthodes de classe

Dans l'exemple VIII.2-1, on a utilisé `-1` pour préciser qu'un âge était inconnu. Si on change cette convention, on devra modifier toutes les occurrences éventuelles de `-1`. Il vaut donc mieux donner un nom à la valeur conventionnelle (par exemple: `ageInconnu`). D'où:

Exemple VIII.7-1: Convention

```
class Personne {
    private int ageInconnu = -1;
    private String prenom;
    private String nom;
    private int age;
    ...
```

Toute méthode de la classe peut modifier `ageInconnu`, ce qui n'a pas de sens. A l'évidence, il s'agit d'une constante. On peut le préciser à l'aide du mot-clé **`final`**. D'où la déclaration:

```
private final int ageInconnu = -1;
```

ce qui empêche toute modification de `ageInconnu`. Toute instance de la classe `Personne` comprendra la constante `ageInconnu`, ce qui est parfaitement inutile. En effet, la constante n'est pas liée à un objet particulier, mais elle est spécifique à la classe elle-même. On peut le préciser à l'aide du mot-clé **`static`**. D'où la déclaration:

```
private static final int ageInconnu = -1;
```

Par analogie, une classe doit pouvoir définir des méthodes (fonctions) qui n'agissent pas sur une de ses instances. Là encore, on peut le préciser à l'aide du même mot-clé **`static`**.

Définitions:

On fera la distinction entre *variables de classe* et *variables d'instance* et, respectivement, *méthodes de classe* et *méthodes d'instance*. Les variables et les méthodes de classes sont introduites à l'aide du mot-clé **`static`**.

Remarque:

Evocation d'une méthode:

- d'instance: `nomObject.methode(...)`
- de classe: `nomClasse.methode(...)`

Exemple (méthode de classe)**:** `Math.cos(...)`

XIX PACKAGE

XIX.1 Définition. Organisation

Le concept de package permet d'organiser les classes en les regroupant dans des ensembles et des sous-ensembles (sous-packages) fonctionnels.

La librairie standard se compose de plusieurs packages. Par exemple, le package `lang` qui contient, entre autres, les classes `Object, String, Integer, System, Math, ...`

Chaque package est contenu dans un répertoire. Les packages de la librairie Java sont situés dans le répertoire `java.` Attention de ne pas confondre la hiérarchie des classes (cf. VIII.5) et l'arborescence des packages.

Chemin d'accès:

```
java.repertoire.sousRepertoire.packages
```

Seules les classes publiques de `java.lang` sont directement utilisables. Dans tous les autres cas, il faut employer le mot réservé **`import`** suivi du chemin d'accès à la classe particulière qu'on veut utiliser.

Exemple IX.1-1 : package util

```
import java.util.Date;
// pour utiliser la classe Date du package util
import java.util.*;
// pour utiliser les classes publiques du package util
```

XIX.2 Modificateur d'accès

Une classe peut être précédée ou non du modificateur d'accès **`public`**.

- **`public class`** `X` ... `X` est accessible en dehors du package qui la contient;
- **`class`** `Y` ... `Y` n'est accessible qu'aux autres classes du package qui la contiennent.

Il existe encore deux autres modificateurs de classe:

- **`final`** la classe précédée de ce mot-clé ne peut pas être sous-classée
- **`abstract`** la classe est incomplète (sera étudié plus tard)..

Exemples IX.2-1 : modificateurs de classe

```
public final class String extends ... {
       etc...
}


public final class Integer extends ... {
       etc...
}
```

XX TABLEAUX

XX.1 Création

Un tableau est une collection de composants tous de même type qu'on accède grâce à un indice de type entier.

Exemple X.1-1 : tableaux avec spécification des bornes

```
int vecteur [ ] = new int [12];
      // un vecteur de 12 entiers
double matrice [ ] [ ] = new double [5] [5];
      // une matrice carrée de 25 réels
String page [ ] = new String [42];
      // une page de 42 lignes
char livre [ ] [ ] [ ] = new char [260] [35] [60];
       /* 260 pages contenant chacune 35 lignes de au plus 60
caractères */
```

Une autre manière de créer un tableau est de lui donner une valeur initiale.

Exemples X.1-2 : tableaux avec spécification de valeurs initiales

```
int v [ ] = {2,4,6}; // un vecteur de 3 entiers
double m [ ] [ ] = {{2.0,4.5},{-6.3,2.1}};
String texte [ ] = {"première ligne",
                    "deuxième ligne",
                    "troisième ligne"};
```

Remarques:
Il existe plusieurs syntaxes pour définir un tableau. Les crochets peuvent suivrel'identificateur de la variable ou l'identificateur du type.

Exemples X.1-3 : différentes syntaxes pour déclarer un tableau

```
int v [ ] = ... ;
     est équivalent à :
int [ ] v = ... ;
```

XX.2 Accès

`identificateur` [`ind 1`] [`ind 2`] ... [`ind n`]

`ind i` : expressions de type entier.

Attention : les éléments sont numérotés à partir de 0 !

Exemples X.2-1 : accès aux éléments d'un tableau

`vecteur [4]` : 5ème élément de `vecteur`
`matrice [i][j]` : élément situé à l'intersection de la (i + 1)ème ligne et (j + 1)ème colonne de `matrice`.
`v [1]` vaut 4
`m [1][0]` vaut -6.3
`texte [3]` est indéfini !

XX.3 Parcours

Pour parcourir les éléments d'un tableau; on utilise, en général, une ou plusieurs "instructions pour" (cf. VII.3).

Exemples X.3-1 : impression de tous les éléments d'un tableau à deux dimensions

```
for (int i=0; i<2; i++)
   for (int j=0; j<2; j++)
      System.out.println (m[i][j]);
```

Annexes

et

Références

bibliographiques

Annexe 1
PRINCIPAUX COMPOSANTS GRAPHIQUES

1) Etiquette

Une étiquette *(Label)* permet d'afficher un texte:

```
Ce programme s'appelle presenteToi
```

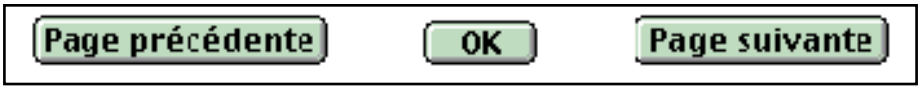
Méthodes:

public `String getText()`
cette méthode retourne, sous forme de chaîne, le texte contenu dans l'étiquette.

public void `setText(String texte)`
cette méthode permet de modifier le texte de l'étiquette.

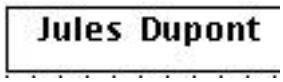
2) Bouton

Un bouton *(Button)* permet de déclencher une action.



3) Champ de texte

Un champ de texte *(Text Field)* permet de saisir un texte.



Méthodes:

public `String getText()`
retourne le texte qui se trouve dans le champ texte.

public void `setText(String t)`
permet de modifier le contenu d'un champ texte.

4) Zone de texte

Une zone de texte *(Text Area)* permet de saisir un texte sur plusieurs lignes.

Méthodes:

`public String getText()`
retourne le texte qui se trouve dans la zone de texte.

`public void setText(String t)`
permet de modifier tout le contenu d'une zone de texte.

`public void append(String t)`
ajoute le texte t à la fin de la zone de texte.

5) Case à cocher

Une case à cocher *(Checkbox)* permet de fixer des choix.

Méthodes:

`public boolean getState()`
retourne `true` si la case est cochée et `false` dans le cas contraire.

`public void setState(boolean b)`
coche ou décoche la case selon la valeur de b.

6) Bouton radio

Un bouton radio *(Radio Button)* permet d'effectuer un choix unique dans un ensemble d'options.

7) Liste

Une liste *(List)* présente plusieurs champs qui peuvent être sélectionnés. Une liste est munie ou non d'une barre de défilement verticale.

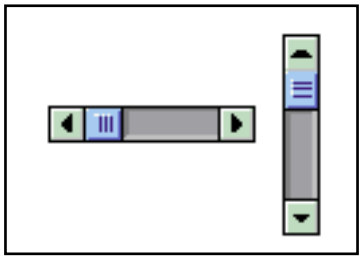
Méthodes:

```
public int [] getSelectedIndexes
```
retourne les indices des éléments sélectionnés sous forme d'un tableau d'entiers.

```
public String[] getSelectedItems()
```
retourne les éléments sélectionnés sous forme d'un tableau de chaînes.

8) Barre de défilement

Une barre de défilement ou glissière *(Scrollbar)* permet de commander le déplacement du contenu d'un composant. Elle permet aussi de choisir une valeur entre un minimum et un maximum. On en trouve de deux types : les horizontales et les verticales.



9) Canevas

Un canevas *(Canvas)* permet de dessiner.

Annexe 2
LA CLASSE STRING

La classe `String` permet de représenter des chaînes de caractères. Tout littéral de chaîne, comme "Hello", est implémenté par une instance de cette classe.

Un objet de la classe `String` est toujours constant, sa valeur ne peut pas être changée après sa création.

Exemple A2.21 La classe String

```
String s = "Hello";
```
est équivalent à :
```
String s;
...
s = new String("Hello");
```

Opérateur

L'opérateur + permet de concaténer des chaînes de caractères.

Exemple A2.22 Concaténation de chaînes de caractères

```
String s ="Hello";
String str;
...
str = s + " World";
```
str vaut : "Hello World"

Remarque :

Lors d'une opération de concaténation, les opérandes de type primitif (entier, réel, caractère et booléen) sont automatiquement convertis en chaînes.

Exemple A2.23 Conversion de types primitifs

```
int i = 124;
double pi = 3.14159;
String s;
...
s = "Résultat: " + i + '*' + pi;
s vaut: Résultat: 124*3.14159
```

Quelques méthodes :

Le premier caractère d'une chaîne porte l'indice 0 !

- `int length()`

  retourne la longueur de la chaîne.

- `char charAt (int index)`

  retourne le caractère qui occupe la position spécifiée par `index`.

- `boolean equals (String str)`

  vrai si la chaîne est égal à `str`.

- `int compareTo (String str)`

  nul: les chaînes sont égales
  négatif: la chaîne précède lexicographiquement son argument
  positif: la chaîne suit lexicographiquement son argument

- `int indexOf (String str)`

  retourne l'indice du premier caractère de `str` dans la chaîne. Si `str` n'est pas une sous-chaîne de la chaîne, la méthode retourne -1.

- `String substring (int beginIndex, int endIndex)`

  retourne la sous-chaîne qui commence par `beginIndex` et se termine par `endIndex-1`.
  On a donc: `s` est identique à `s.substring (0, s.length()).`

- `String toLowerCase()`

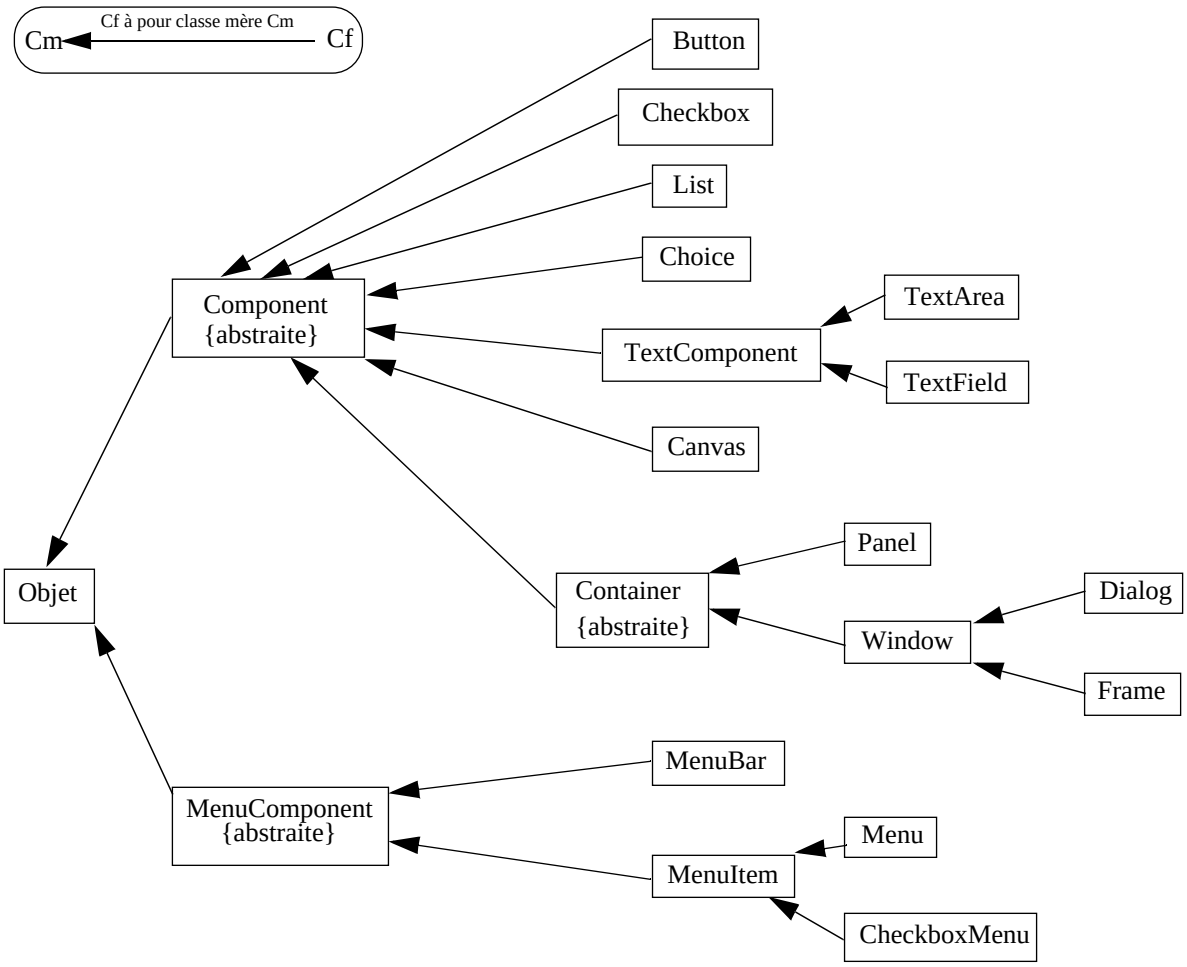
  transforme la chaîne en minuscules.

- `String toUpperCase()`
  transforme la chaîne en majuscules.

Annexe 3
QUELQUES CARACTERES SPECIAUX

Caractères graphiques	Signification
°	degré
§	paragraphe
"	guillemet
%	pourcentage
&	et (commercial)
/	slash
\	backslash [<maj><alt>/]
@	at sign, a commercial, "queue de singe" [<alt>g]
#	dièse [<alt>3]
$	dollar
£	pound, "livre sterling"
~	tilde [<alt>n]
[ ]	crochets [<alt>5 et <alt>6]
{ }	accolades [<alt>8 et <alt>9]
caractères non graphiques	
	retour de chariot
→|	tabulateur
←	backspace, touche effacer
⌘	"touche pomme"
<esc>	escape - sortir, s'échapper
<ctrl>	control - contrôler
<alt>	touche alt (alternative)

Annexe 4
Hiérarchie partielle des classes dans AWT[1]



1) AWT : Abstract Window Toolkit

Annexe 5: QUELQUES CLASSES EN JAVA

StringTokenizer

La classe StringTokenizer permet de découper en morceaux une chaîne de caractères en fonction d'un certain nombre de caractères de séparation.

Constructeur:

`StringTokenizer (String chaine_a_découper,String caracteres_de_separation)`

Ce constructeur a deux arguments: la chaîne que l'on désire découper et le ou les caractère(s) que l'on emploie pour le découpage.

Méthodes:

`int` `countTokens()`
cette méthode retourne le nombre de morceaux détecté dans la chaîne.

`String nextToken()`
cette méthode retourne le prochain morceau.

`boolean` `hasMoreTokens()`
cette méthode retourne vrai s'il reste des morceaux à retourner, faux sinon.

Exemple:

On dispose d'une chaîne contenant les jours de la semaine séparés par des barres obliques:

```
String chaine="lundi/mardi/mercredi/jeudi/vendredi";
```

On désire afficher dans la console le nombre de jours ainsi que les jours. On commence par découper la chaîne à l'aide d'un `StringTokenizer`:

```
StringTokenizer stTo = new StringTokenizer (chaine,"/");
```

On affiche le nombre de jours à l'aide de la méthode `countTokens()`:

```
System.out.println("il y a "+stTo.countTokens()+" jours");
```

Puis à l'aide d'une boucle `while` qui s'exécute tant qu'il reste des éléments, on affiche les jours:

```
while (stTo.hasMoreTokens()){
       System.out.println(stTo.nextToken());
}
```

Ce programme affiche dans la console le texte suivant:

```
il y a 5 jours
lundi
mardi
mercredi
jeudi
vendredi
```

Math

Cette classe permet de manipuler des nombres. Elle contient de nombreuses méthodes très utiles pour effectuer des calculs avec des nombres entiers et réels.

Cette classe se trouve dans le paquetage java.lang.

Constantes

public final static double E
E représente une approximation de la base des logarithmes naturels (2.71828183).

public final static double PI
PI représente une approximation du nombre irrationnel pi (3.14159265).

Méthodes:

public static double pow**(double** a, **double** b)
retourne sous la forme d'un double le premier argument élevé à la puissance du second.

Exemple:

```
Math.pow(5,2)=25
```

public static int round(**float** a)
cette méthode arrondit le nombre réel a.

public static double random()
retourne un double aléatoire compris entre 0.0 et 1.0.

Exemple:

Si l'on désire obtenir un entier entre 0 et 10, il faut prendre un nombre aléatoire, le multiplier par 10 et l'arrondir:

```
int nombre_entre_0_et_10=(int)Math.round(Math.random()*10);
```

Integer

Cette classe se trouve dans le paquetage java.lang. Elle permet de traiter des nombres entiers comme des objets et non comme des types simples. Dans la pratique, on l'utilise surtout pour convertir des chaînes de caractères en nombres entiers.

Constantes:

`public static final int` `MAX_VALUE`
représente la valeur maximale qui peut être stockée dans une variable entière (2'147'483'647).

`public static final int` `MIN_VALUE`
représente la valeur minimale qui peut être stockée dans une variable entière (-2'147'483'648).

Méthode

`public static int` `parseInt (String s)`
convertit la chaîne s en un entier **`int`**. Convertit par exemple la chaîne "347" en le nombre 347 qui sont des choses différentes. On l'emploie par exemple lorsque l'on veut récupérer un nombre entré par un utilisateur dans un champ de texte (cf. Annexe 1).

Graphics

La classe Graphics permet de dessiner sur l'applet ou sur un des divers composants graphiques.

Constructeur

On n'emploie pas de constructeur pour déclarer un objet de type Graphics. A la place, on emploie la méthode `getGraphics()` qui permet de récupérer, à partir d'un composant donné, un objet graphique sur lequel on peut dessiner.

```
Graphics g = canvas1.getGraphics();
```

Méthodes:

`void` `drawLine(`**`int`** `x1, `**`int`** `y1, `**`int`** `x2, `**`int`** `y2)`

cette méthode permet de dessiner une ligne du point (x1,y1) au point (x2,y2).

`void` `drawRect (`**`int`** `x, `**`int`** `y, `**`int`** `largeur, `**`int`** `hauteur)`

cette méthode permet de dessiner un rectangle défini par les coordonnées de son sommet supérieur gauche et par sa largeur et sa hauteur.

`void` `drawOval (`**`int`** `x, `**`int`** `y, `**`int`** `largeur, `**`int`** `hauteur)`

cette méthode permet de dessiner un ovale. Cet ovale est défini par le rectangle qui le contient. Les paramètres servent donc à définir le rectangle, comme dans la méthode `drawRect`.

`void` `setColor(Color c)`

cette méthode permet de changer la couleur courante. Cette couleur sera employée pour tous les dessins faits dans l'objet graphique. Par défaut la couleur est noire.

Exemple:

```
g.setColor(Color.pink)
```

sélectionne la couleur rose pour dessiner dans l'objet graphique g.

JAVA - Livres de référence

• HANSEN Brinch,

`Programming for Everyone in Java`
New York: Springer-Verlag, 1999 (54.- CHF)

• CAMPIONE Mary, WALRATH Kathy,

`The Java Tutorial Second Edition Object-Oriented Programming for the Internet`
Second Edition, Reading, MA: Addison-Wesley, 1998

• DEITEL H.M., DEITEL P.J.,

`Java How to Program: 2nd Edition`
Upper Saddle River: Prentice-Hall, 1998 (~ 90.- CHF)

• CLAVEL G., MIROUZE N., PICHON E., SOUKAL M.

`Java: La Synthèse`
2ème éd., Paris: Masson, 1998 (59.20 CHF)

• FLANAGAN David,

`Java in a Nutshell: 2nd Edition`
Sebastopol, CA: O'Reilly, 1997 (50.20 CHF)

• LEMAY Laura, PERKINS Charles L.,

`Apprenez Java 1.1 en 21 Jours`
Paris: Simon & Schuster Macmillan, 1997

• FRIEDEL David, KERIEVSKY Joshua, POOTS Anthony, RODLEY John,

`Symantec Café: Outils de Développement Visuel pour Java`
Paris: International Thomson Publishing, 1997 (82.80 CHF)

• ARNOLD K., GOSLING J.

`Le Langage Java`
Paris: Thomson Publishing, 1996 (78.30 CHF)

• GOSLING J., JOY B., STEELE G.

`The Java Language Specification`
Reading, MA, Addison Wesley, 1996 (~ 70.- CHF)